Using the X Window System

HP 9000 Series 300/800 Computers

HP Part Number 98794-90001

Hewlett-Packard Company
1000 NE Circle Blvd., Corvallis OR 97330

Notice

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MANUAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Certification of conformance with the OSF/Motif user environment is pending.

Printing History

The manual printing date and part number indicate its current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. The manual part number will change when extensive changes are made.

Manual updates may be issued between editions to correct errors or document product changes. To ensure that you receive these updates or new editions, you should subscribe to the appropriate product support service. See your HP sales representative for details.

December 1988 ... Edition 2
September 1989 ... Edition 3

Contents

1

How to Improve Your X Life

Welcome to graphical user interfaces ("windows") and to the X Window System version 11 (X11 or X) in particular. In this chapter you'll find out how this manual is organized and some of the conventions it uses. You'll also find some tips to make learning about X11 easier and to improve your X life thereafter.

How This Manual Is Organized

This manual is organized so that the less technical information comes first.

If you're new to computers, new to HP-UX, or have had some window experience—but never this much control of your screen environment—you'll want to read this chapter and chapters 2 and 3. You'll also find the glossary and the index helpful.

Chapter 1	Introduces this manual and gives some tips on HP-UX and networking.
Chapter 2	Explains the window environment and sets the stage for chapter 3.
Chapter 3	Provides a beginner-level introduction to using the X Window System.

If you're a system administrator or programmer—someone familiar with computers and how they operate—you'll probably be more interested in the more technical information in the second half of this manual.

Chapter 4	Explains how to run programs from the command line.
Chapter 5	Discusses customizing the X environment to suit your personal needs or the needs of the users who use your system.

Chapter 6	Offers a detailed explanation of the OSF/Motif Window Manager.
Chapter 7	Provides information about customizing special X environments.
Chapter 8	Discusses printing and screen dumping.
Chapter 9	Discusses the use of Starbase graphics.
Appendix A	Discusses other window managers.
Reference	Contains "man" pages—the definitive description—for current X clients.

But please, System Administrator, don't just skim the man pages, tweak the `sys.Xdefaults` and `system.mwmrc` files, and then bury this manual on a bookshelf. Your users, the people who depend upon you for support, could use this manual to make life a little easier for themselves—and for you. Make it available to them and encourage them to read it. As they become self-sufficient within their window environment, your support tasks become easier.

Ultimately, whether you're a new user or an experienced user, the purpose of this manual is to improve your X life.

Conventions

As you read this manual, notice the following typographical conventions:

Table 1-1. Typographical Conventions

If you see ...	It means ...
`computer text`	This text is displayed by the computer or text that you type exactly as shown. For example, `login:` is a login prompt displayed by the computer.
italic text	A book title, emphasized text, or text that you supply. For example, hpterm -fg *color* means you type "hpterm -fg" followed by a color you choose.
	You press the corresponding key on the keyboard. For example, CTRL Left Shift Reset means you hold down the CTRL key, the Left Shift key, and the Reset all at the same time.
`[ ]`	An optional parameter that can be left off if you don't need that functionality. For example, `xload [-rv] &` means that you must type "xload" but don't have to type "-rv".
`{ }`	A list containing *mutually exclusive* optional parameters. For example, `xset r { on / off }` means that option `r` can be set to either `on` or `off`, but not both.
bold text	The definition of this term follows. Additionally, the term is defined in the glossary.

Also, you can use the X Window System with either a two- or a three-button mouse by observing the following conventions:

Table 1-2. Mouse Buttons and Their Locations

If you see ...	On a 2-button mouse press ...	On a 3-button mouse press ...
Button 1	The left button.	The left button.
Button 2	Both buttons simultaneously	The middle button.
Button 3	The right button.	The right button.

System Administrators, these are the default mouse button settings and can be changed as described in chapter 7.

Running HP-UX: Some Tips

If you are new to HP-UX and to the X Window System, take heart: You're not alone. A wide variety of users, many just like yourself, are currently learning HP-UX and X11. The next several paragraphs provide you with information and tools to facilitate the initial stages of learning.

What Is HP-UX and the X Window System?

HP-UX is Hewlett-Packard's implementation of the UNIX operating system. The operating system is the software that controls the operation of the computer system. HP-UX is a multi-user, multi-tasking environment. A multi-user environment means more than one user can be on the system at the same time. A multi-tasking environment means that each of those users can run more than one program at a time.

The X Window System is a window environment. It turns your screen into a "root window" or "desktop" on which you can display smaller windows, each one the equivalent of a full-sized display terminal. Within the X11 environment you can run multiple tasks, viewing their progress in separate windows.

Why Background Processing Is Important

Your programs can run as either foreground or background processes. In any X11 terminal widow, you can only run one program at a time as a foreground process, but you can run many programs as background processes. To run a program as a background process, add an ampersand (`&`) to the end of the command line that starts the program. The ampersand tells the system that the program should be run in the background. This leaves the foreground free for you to issue more commands.

Take, for example, the following command:

```
xclock &
```

This command starts a clock. The `&` tells the system to display the clock, but as a background process, so you can use the foreground to enter more commands. Without the `&`, the clock would still display, but in the foreground. The window from which you issued the command would ignore everything else, including your keyboard commands, as long as the clock remained the foreground process. This could prove inconvenient, even to inveterate clock-watchers.

If you forget an `&`, you will need to stop that program to regain control of the foreground—a task not always easy to accomplish (see either "Exiting from the X Window System" in chapter 3 or "Stopping Programs" in chapter 4).

One last note on foreground and background: Don't confuse foreground and background *processing* with foreground and background *color.* The foreground and background that you process are not the same foreground and background that you color. Foreground and background processes are activities of the computer; the foreground and background colors are graphical elements that display on the screen.

Case Sensitivity and Other Typographical Tips

HP-UX distinguishes between uppercase and lowercase letters. A file named `.xdefaults` is *not* the same file as `.Xdefaults`. Use uppercase letters where indicated and *only* where indicated.

Also, the number "1" (one) looks like a lowercase "l" (el) to our human eyes. The system, however, can readily distinguish the difference and often seems to do so with a vengeance.

Don't confuse the "0" (zero) with the upper case "O" (oh) for the same reason.

White space (extra spaces or tabs) at the end of a command line in a text file sometimes alters the meaning of the command. Files such as `.rhosts` are especially vulnerable. After modifying a file, check for unwanted white space.

And finally, *watch your spelling.*

Working with HP-UX Manuals

HP-UX manuals typically have a section devoted to reference information. This section contains "man" (manual) pages that provide specific information about a command, function, or program. The man page is the most definitive source of information. You will find man pages in the reference section at the back of this manual.

Logging In to HP-UX

Most HP-UX systems require you to log into a system before you gain access to the resources available on that system. The administrator of the system must provide you with a login account. When you have a login, you will be able to log into that system by providing your login name and your personal password. When you are logged in, you may use the resources available such as the X Window System.

Note that on some systems the system administrator may have configured your login process so that you automatically start your X environment.

For More Information

Several beginner's guides come with your computer system.

Table 1-3. Beginner's Guides

To learn about ...	Look through this guide ...	HP Part Number
Using the HP-UX operating system concepts and commands.	*A Beginner's Guide to HP-UX*	98594-90006
Using shells to increase performance.	*A Beginner's Guide to Using Shells*	98594-90008
Editing commands for the `vi` editor.	*A Beginner's Guide to Text Editing*	98594-90010
Customizing your own X Window System environment.	*A Beginner's Guide to the X Window System*	98594-90002

If you are new to the system, taking the time to study these guides will help clarify questions you may have.

There is also a great deal of information available about the HP-UX operating system in the *HP-UX Reference* volumes that accompany the operating system.

Additionally, information about programming in the X Window System environment is available in the following manuals:

Table 1-4. Reference Manuals

To learn about ...	Look through this manual ...	HP Part Number
Writing and using widgets in application programs.	*Programming with the Xt Intrinsics* *HP OSF/Motif Programmer's Guide*	98794-90008 98794-90005
Fortran Bindings and Native Language I/O systems.	*Programming with Xlib*	98794-90002
Writing graphics programs for X.	*Programming with Xlib*	98794-90002

These manuals contain information about the HP OSF/Motif user environment.

- *HP OSF/Motif Programmer's Guide.*

- *HP OSF/Motif Programmer's Reference.*
- *HP OSF/Motif Style Guide.*

Finally, depending on your needs, the following books about the X Window System might prove useful:

- *Introduction to the X Window System* by Oliver Jones. Prentice Hall, Englewood Cliffs, NJ:1989.
- *Xlib Programming Manual for Version 11* by Adrian Nye. O'Reilly and Associates, Newton, MA:1988.
- *Xlib Reference Manual for Version 11* edited by Adrian Nye. O'Reilly and Associates, Newton, MA:1988.
- *X Window System User's Guide* by Tim O'Reilly, Valerie Quercia, and Linda Lamb. O'Reilly and Associates, Newton, MA:1988.
- *X Window Systems Programming and Applications with Xt* by Douglas A. Young. Prentice Hall, Englewook Cliffs, NJ:1989.

Where to Go Next

Now that you've finished these preliminaries, you have a choice. If you feel comfortable with (or aren't interested in an explanation of) graphical user interfaces, skip chapter 2 and read chapter 3 on how to use the X Window System.

If you've had some experience with graphical user interfaces and the X Window System in particular, you might want to skip all the way to chapters 4 and 5 to find out how to run X11 clients and customize your X11 environment to your individual needs.

2

Understanding Window Systems

This chapter is written for new users. If you're not familiar with HP-UX or window environments, this chapter's for you. It describes the following key elements:

- Basic window concepts.
- A typical X Window System environment.
- An example of a distributed computing environment.

This chapter demonstrates the power and the flexibility of the X Window System.

What Is the X Window System?

The X Window System is a **graphical user interface,** a way of communicating with your computer using visual images (graphics). You can better understand the importance of the X Window System and why its possibilities are so exciting if you compare it to the "traditional" user interface, the command-line prompt.

In contrast to the austerity of the command-line prompt, the X Window System offers a visually rich connection to your computer. This connection, the user interface, is characterized by easily recognizable graphical features: windows, selection menus, and icons.

X11 surrounds your interaction with the computer system in a visual metaphor more intuitively meaningful—especially to novice users—than the command-line prompt with its often esoteric commands and obscure parameters, for example, "pushing" a button. X11 provides you with a friendly, easy-to-use work environment.

X11 Is Based on the Server-Client Interaction Model

The X Window System is based on a server-client interaction model.

The **server** is really what you "start" when you "start X11." The server controls all access to input devices (typically a mouse and keyboard) and all access to output devices (typically a display screen). You can visualize its position in the scheme of things by thinking of it as standing between the programs you run on your system and your system's input and display devices.

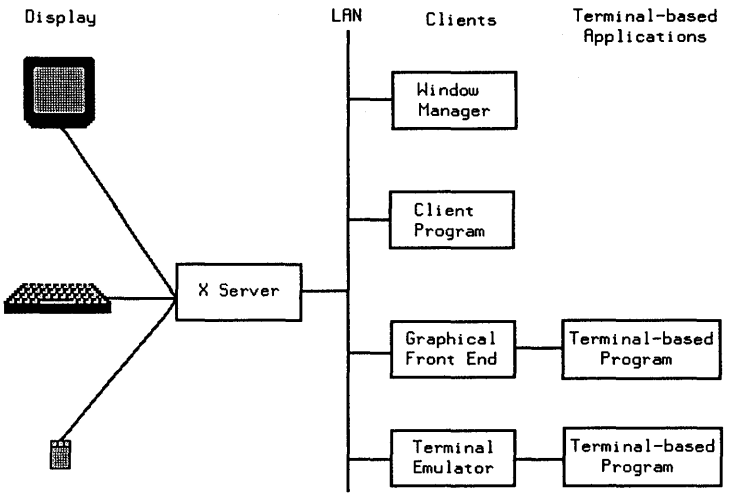


Figure 2-1. The Server Controls Display Access

A **client** is any program written especially to run with the server. Another way of looking at it is to view the client as "window smart." Clients know about windows and how to make use of them. All other programs are **non-clients,** programs that don't know how to make use of windows.

Multi-Tasking Makes X11 a Powerful Tool

Part of the X Window System's power comes from the computer system's **multi-tasking** ability. Multi-tasking is the ability to execute several programs simultaneously. Each program is a separate task (process). The X Window System brings multi-tasking out of the realm of the power user and into the hands of the novice user in search of increased efficiency. In your X11 environment, you run each program in a separate window as a separate process. Windows may overlap on the screen, but their processes don't interfere with each other.

For example, you could have the system recalculate a large spreadsheet in one window while you shift your attention between editing a monthly report in a second window and answering your electronic mail in a third. Each program normally has a main window for visual interaction, and each window has its own input and output.

You focus your attention on a particular window by moving the mouse pointer into that window and pressing button 1. The window thus pointed to becomes the **active window.** While you focus on one window, other windows continue running unattended or wait for your input.

Multi-tasking is possible in part because of the way the computer system divides all processes into **foreground processes** and **background processes.** Background processes are the ones that run unattended or wait until they get your input. You can have as many background processes running in a window as you like. A foreground process is the process that has the window's attention at the moment. You can have only one foreground process running in each window.

The **ampersand (&)** at the end of a command line initiates background processing.

X Allows Both Local and Remote Access

Any computing environment allows you **local access,** the ability to run a program on the computer in front of which you're sitting. Networked computing environments also allow you **remote access** to programs, the ability to run a program on a computer *other* than the one at which you're sitting.

Using the X Window System, you can run programs both locally and remotely at the same time. You also have greater control over where the output displays. If you wish, you can run a program locally and display the output on the screen of a remote system; or the opposite, run a program remotely and display the output in a window on your screen; or run a program remotely and have it display on yet another remote screen.

The X Window System Allows Multi-Vendor Networking

A final feature of X11 worth mentioning is the X Window System's acceptance as an industry standard for UNIX operating system network protocol. Since all X Window System hardware and software vendors communicate using the X protocol, programs from different vendors can be run remotely and be viewed on your local system. Thus, computer networks composed of hardware and software from multiple vendors, instead of being a "nightmare of incompatibility," become powerful resources for specialized applications, allowing the user to select the best hardware and software for the application without compromising performance for compatibility.

The Parts of a Typical X Window System

Your personal window environment can be relatively simple or rather elaborate. The details depend on your personal computing needs, the programs you use, and how you customize three X Window System configuration files. However, all X Window System environments have the following features in common:

- Computer hardware (a system) on which to run the software.
- An X server program to control communication between the display and client programs.
- A window manager to control the display's window environment.
- Application programs to provide useful services.

The Computer Hardware System

The hardware system consists of several components:

- System Processing Unit (SPU).
- Hard disk.
- Keyboard.
- Mouse, or other pointing device.
- Display screen.
- Connection to a Local Area Network (LAN).

The SPU Does the Computing

The System Processing Unit or SPU is the "brains" of the computer. The SPU contains the logic circuitry, which is driven by the software and performs all the processing that takes place. The SPU of your system runs the X server that provides your window environment, takes care of foreground and background processing, and controls local and remote accessing of your system's resources. Using X, you can run programs that are stored on your own hard disk (local processing) or that are stored on someone else's hard disk using their SPU (remote processing).

The Hard Disk Stores Data

The hard disk stores programs and data files. No processing takes place on the hard disk, only storage. Some HP 9000 Series 300 and Series 800 configurations are called **diskless clusters** because groups of users share the same hard disk.

The Keyboard Enters Text

The keyboard is an **input device,** a device used to type information into the computer. This information could be the text of a letter or the next command that the computer should execute, depending on whether you type the text into a file or on the command line.

Although the keyboard is frequently used in conjunction with a mouse, it does not need to be. You can configure your X11 environment so that you can use the keyboard for both text entry (its usual purpose) and for pointing and

selecting (the mouse's usual purpose). For example, this **mouseless operation** would be beneficial in any situation where desk space was at a premium.

The Pointing Device (Mouse) Points and Selects

The keyboard enters characters; a pointing device points and selects. A mouse is the most used pointing device. Sliding the mouse on your desktop moves the **pointer,** the current screen location of the mouse, on the screen. Using the mouse, you can point to an object on the screen, for instance a window, and select an action to perform, such as resizing. Selection is made by pressing button 1 on the mouse. As mentioned, however, mouse movements and button presses can be associated with keyboard key presses for mouseless operation.

The Screen Displays Output

The principal output device for the X Window System environment is the **display.** A typical display consists of one physical screen per mouse and keyboard. However, depending on the specialized nature of the application, a display may include as many as four physical screens, all using the same mouse and keyboard.

The **screen** is the physical CRT (Cathode Ray Tube) that displays what you type on the keyboard. The screen also shows you the position of the pointer and windows, and provides you with visible indications of the status of executing programs.

Conceptually, the screen becomes the **root window** when you start the X Window System. The root window contains all the windows, menus, and icons that compose the visual elements of your X11 environment.

Technically, the screen is known as a **bitmapped device** because the graphical elements (windows and icons) that it displays are stored by the computer as a **bitmap,** a pattern of bits (dots) that can be readily displayed as graphical images.

The LAN Connects to the Network

The LAN is composed of hardware and software. The hardware part connects your computer system physically (using a cable) to a network that includes other computer systems at your site and could encompass other networks

at different locations. The LAN enables you to take advantage of remote processing capabilities of X.

Other Pointing Devices

Although the mouse is the most common pointing device, the X Window System display server (the program that "runs" X on your system) supports other HP-HIL (Hewlett-Packard Human Interface Link) pointing devices, for example a digitizer tablet or track ball. References in this manual to mouse actions apply also to corresponding actions with other HP-HIL pointing devices.

The X Server Controls Communication

The **server** is the program that controls the screen, keyboard, and mouse, and processes all communication requests. The X server is really what runs when you "run X11." The server updates the windows on the screen as a client generates new information or as you enter information through an input device. All client programs communicate through the server.

Because the server controls communication with the display screen, it is sometimes called the **display server.** Either name is correct.

The Window Manager Controls Your Windows

The window manager is your main means of dynamically controlling the size, shape, state (icon or normal), and location of the windows on your screen. Several different window managers exist; the window manager for the Hewlett-Packard implementation of the X Window System is called the OSF/Motif Window Manager (`mwm`). The window manager includes:

- menus
- icons
- window frames

The Window and Root Menus

One way that you can control the operation of your window environment is by choosing an action from a **menu.** A menu is a indow that contains a list

of selections—exactly like a restaurant menu. The window manager has two menus:

Window menu	One for each window on your screen. A window menu controls the particular window to which it is attached.
Root window menu	The menu for the root window. The root menu controls actions that are generic and refer to no particular window.

The following figure shows a window with the window menu displayed and the "maximize" selection highlighted.

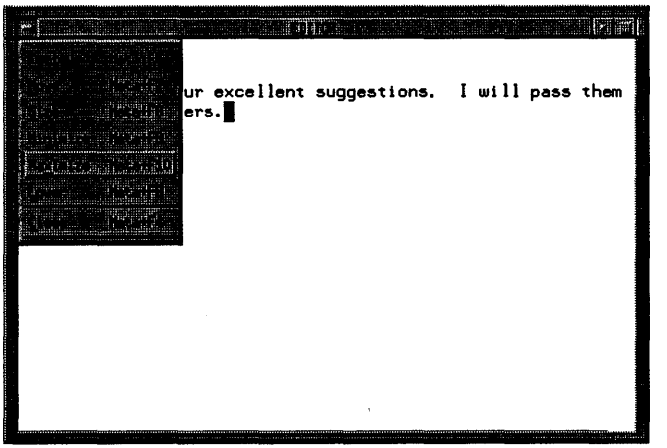


Figure 2-2. The Window Menu with "Maximize" Selected

You can configure your window manager to make life easier for yourself. For example, you can add a selection to the root menu that enables you to log onto a remote host and run an application automatically. You can also create submenus of related activities. One popular submenu is a list of remote hosts to log onto. Chapter 6 of this manual discusses configuring the OSF/Motif Window Manager. Appendix A discusses other window managers, hpwm and uwm, which were used in previous releases of the X Window System.

Icons

Because your display will often contain several windows, you may find it convenient to set aside a window you're not currently using *without stopping the processing in that window.* You do this by changing the window into an **icon,** a small, easily identifiable graphic symbol that represents the window but takes little space on the screen.

The contents of an iconified window aren't visible. But you can quickly convert the icon to its original window representation whenever you wish to use the window again. Any processing that was occurring in the window as it was iconified continues as long as it doesn't require additional input from you. You won't be able to see output or enter input until you change the icon back into a window.

The figure below shows several icons, each representing a different type of client.

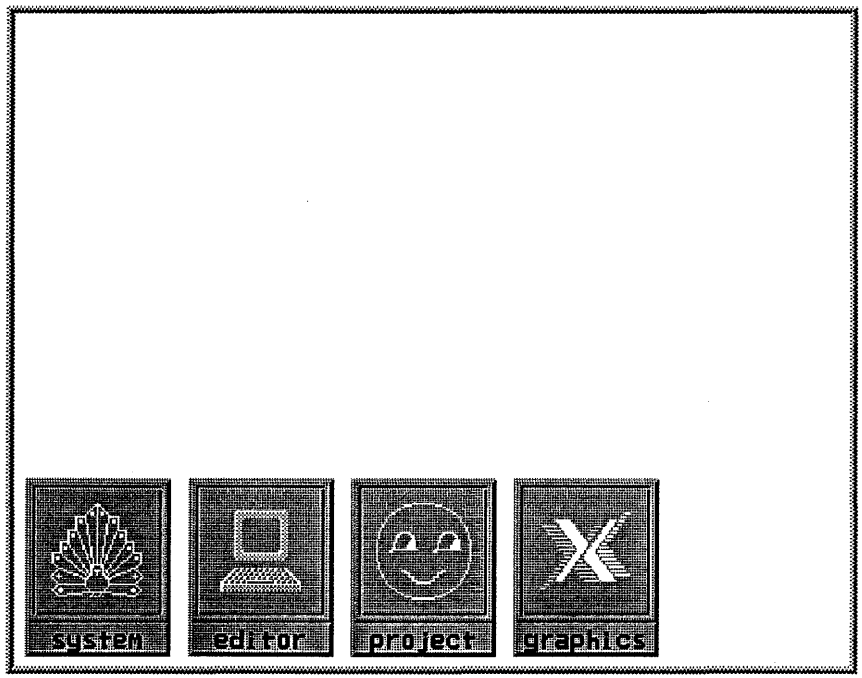


Figure 2-3. Icons Replace Windows Giving You More Room

Window Frame Decoration

The window manager provides a functional frame around each window in the root window. The frame, sometimes called **window decoration,** consists of graphical control devices that enable you to display the window menu, maximize or iconify the window, or move and resize the window.

Application Programs Run in Your X Environment

An **application program** is a computer program that performs some useful function like word processing or data base management. The applications you run while you use X may be stored on the hard disk attached to your system or on the hard disk of a remote system. The X11 server communicates with application programs just as easily over the LAN as locally.

You can sort all application programs into two categories:

- Those that know about windows and incorporate windowing behavior into their own behavior (client programs).
- Those that don't know about windows and think that they must always be running on a separate terminal (non-client programs).

Window-Smart Programs Are Called Clients

A **client** is a program written especially for the X Window System. Clients are referred to as **window-based** programs. The window manager that controls the windows on your screen is a client. The windows themselves are clients. Clients are "smart" enough to create their own windows if they need to display output. Note, however, that not all clients create windows. Some clients (like `xwininfo` and `xmodmap`) are content to use an existing terminal emulation window in which to display their output.

Terminal-Based Programs Must Be Fooled

Non-client programs know nothing about windows. They are designed to run alone on display screens or "terminals" and are, therefore, referred to as **terminal-based** programs. Terminal-based programs must have windows created for them so that they can run in a window environment. They are thus "fooled" into operating in the window environment.

You can operate terminal-based programs in the X Window System by using a client program called a **terminal emulator** to provide a window. You start the non-client program in that window. The terminal emulator "fools" the non-client into thinking that it is running on a "real" terminal instead of a window imitating a terminal. This has led some people to describe non-client programs as "window dumb."

The X Window System provides two terminal-emulator clients: `hpterm` and `xterm`. When either is run, it creates a window to emulate a display terminal. A terminal-based program runs happily in this window, acting exactly as if running on a terminal.

The following diagram shows the components of a system running X.

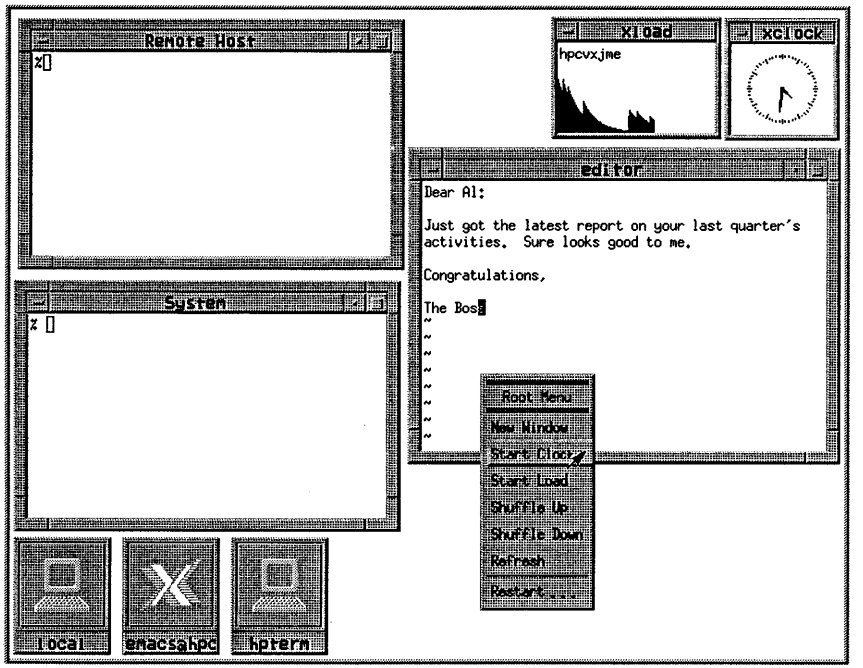


Figure 2-4. Typical Components of an X Window System

The Distributed Computing Environment

A Distributed Computing Environment (DCE) is a group of computer systems joined together into a network. Resources resident on one system are available to all systems. As mentioned earlier, a system that uses X11 is usually connected to a LAN. The LAN provides the link to programs that are resident on physically separate (remote) systems.

X11 really doesn't care where a program is—it simply communicates to the program via the LAN connection. This structure permits you to operate S300 and S800 systems at a strictly local level with all client programs residing locally, or at a networked level with some programs running at the local level while others run on remote systems.

In addition, another system on the LAN can run programs that reside on your S300 or S800 and direct the visual output to *any* screen on the network.

A distributed computing environment, in other words, enables the best possible allocation of processing resources within the existing hardware environment.

The figure below shows a distributed computing environment that provides a number of resources to users who are connected to the LAN and running X.

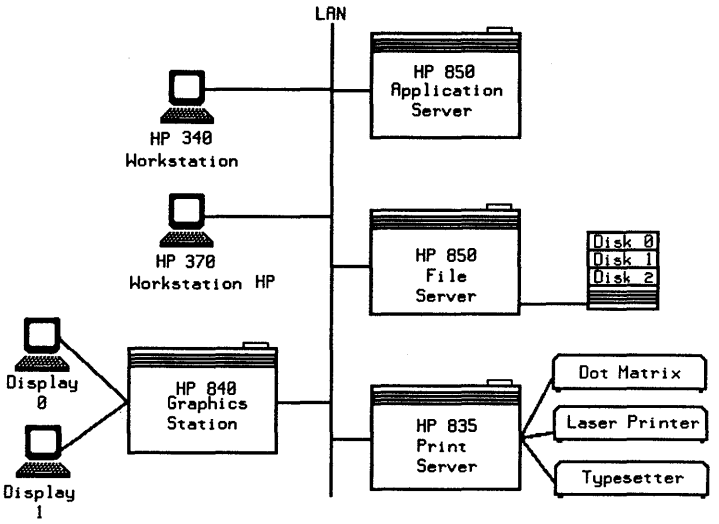


Figure 2-5. A Typical Distributed Computing Environment

As the figure indicates, if you use a system running X11 and connected to a LAN, you have a multitude of resources available. The following sections provide a practical example of how the above environment and the resources contained therein could be used.

Workstations Provide Local and Remote Processing

Two workstations are pictured in the figure, an HP 9000 Series 330 and an an HP 9000 Series 350.

Both workstations can use clients that reside either locally, on their own hard disks, or remotely, on the hard disk of another system (for example the Series 850 application server.) The workstations illustrate the capability of a single system to operate either locally or remotely.

Application Servers Handle Process-Intensive Applications

One of the HP 850s shown in the figure is an **application server.** An application server is a computer that provides the processing power and memory necessary to run large, processor-intensive applications.

A typical user of such an application would be Hank, who works for a large oil company. Hank is currently involved in the search for new oil resources in Alaska. Many variables are considered in the attempt to locate potential oil fields. Hank uses a simulation program that mathematically manipulates all of these variables to produce data that indicates the potential for a certain area. These computations require a tremendous amount of memory, disk space, and processor time.

With a distributed computing environment, Hank can sit at his desk and use his personal workstation to log into the HP 850 application server and enter the necessary data. The actual simulation program and the necessary data files reside on the HP 850. Hank runs the simulation using the processing power of the application server. He has the output directed to a window on his workstation while he is busy performing tasks locally in other windows until the necessary simulation information is available.

Hank is only one of many employees to take advantage of the processing power of the application server. Other employees in the same department or even in a different building can also log in and use the system.

File Servers Supply Data Storage

The other HP 850 shown in the figure is a **file server.** A file server is a computer that controls the storage and retrieval of data from hard disks. A file server means less storage space is required on an individual's local computer. It also provides a relatively inexpensive and quick backup facility.

Let's say Alex is a writer who is responsible for the content of several chapters of a large manual. She works at her desk using an HP 330 as a writer's station and at any given time is working on one of several different projects that total 10 to 15 megabytes of storage on a hard disk. Using the HP 850 file server, she could store her files on a master disk drive and check out the chapters she needs to work on. This leaves a backup of the files on the file server. The file

server can thus be used to maintain current backups by transferring updated files to it on a regular basis.

At any given time, Alex will only have a few chapters stored on her own disk drive; those chapters she is currently working on. If she finds that she needs a copy of another chapter that is not currently residing on her disk, she requires only moments to transfer a copy from the main disk.

Another use for a file server is to serve as a hub for diskless workstations. You can have a cluster of several diskless workstations connected to a single hub with a large disk. Each workstation, or **node,** needs a certain amount of individual space on the disk, but all nodes can share the system and application software, eliminating the need for local system storage and thus saving a considerable amount in overall storage requirements.

Print Servers Control the Printers

The HP 835 shown in the figure has several printers attached to it and acts as a **print server,** a computer that controls spooling and other printing operations. Page formatters and page composition programs reside on the print server and are invoked with the proper commands. When you need a document printed using a particular type of printer, you send it to the print server with the appropriate instructions, and the task is accomplished. This permits a large number of individuals from anywhere in the distributed computing environment to efficiently share printer resources.

If Alex needs a copy of a chapter quickly printed for an immediate review, she instructs the HP 835 to print the chapter using the fast dot-matrix printer. If she needs a letter-quality copy of a document containing elaborate graphics, she routes the letter to the laser printer. For those manuals that need to be typeset, the print server can also drive a typesetter. Alex would again simply direct the appropriate command to the print server to run the document through the typesetter.

Graphics Station for Specialized Graphics Applications

Certain applications are designed to take advantage of graphics accelerators in order to speed up the presentation of graphics on the screen. Generally, engineers working with CAD (Computer-Aided Design) applications are the major users of graphics accelerators. Hewlett-Packard supports graphics acceleration with a graphics library called Starbase. The HP 840 shown in the figure has two high-performance graphics subsystems attached to it. Each subsystem is powerful enough to run the X Window System while running a Starbase application.

Anne is an engineer who is working on a project involving the design of a new, high-speed sailboat hull. The CAD program she uses is very expensive and requires graphics acceleration to accomplish complex shading. When Anne wants to work with the program, she can move to the graphics station where she can use multiple windows provided by X with the CAD program running in one of the windows.

The graphics station permits a larger number of people to share the expensive hardware and software resources required by a CAD/CAM station. Tasks that engineers may have that do not require graphics acceleration can be accomplished at their desks on a more typical workstation.

Multi-Vendor Communications

Another advantage of DCE is its ability to allow computers manufactured by different vendors, running different operating systems, to communicate with each other over the LAN. If you are using a computer made by Hewlett-Packard, you can communicate over the LAN directly with a computer made by Sun, DEC, IBM, or a variety of other manufacturers supporting X, as long as each is running the X Window System and connected to a LAN using the Ethernet protocol standard.

The diagram below shows a multi-vendor environment of computers running different operating systems. Communication over the LAN is a simple task as long as they are all running X11.

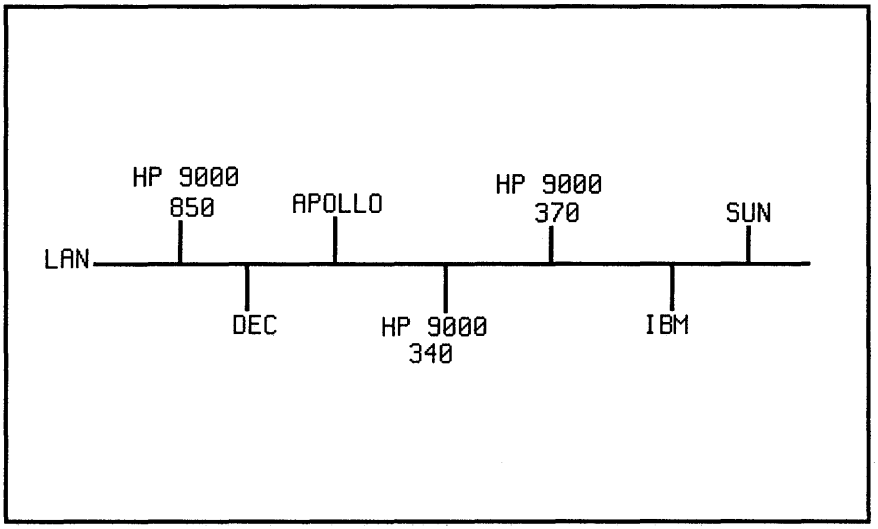


Figure 2-6. Multi-Vendor Communication Is A Benefit of X11 DCE

Where to Go Next

You should continue to chapter 3 to learn how to use your X Window System. Chapter 4 contains information about running client programs from the command line, while the chapters following contain information on customizing your window system environment.

3

Using the X Window System

This chapter covers the basics of window operation. It shows you how to use X once it's been installed on your system. You'll learn how to perform the following tasks:

- Start the X Window System.
- Create, move, resize, and "shuffle" windows.
- Iconify a window and normalize an icon.
- Display menus and make selections.
- Stop programs and correctly exit your X environment.

Starting the X Window System

Before you start the X Window System, you must be logged in to your computer system. Log in using your normal procedure.

Note

The X Window System can't run on a system that's *already* running HP Windows/9000. If you are running HP Windows/9000, you must exit from that window system *before* you start X. (HP Windows/9000 can be installed on your system; it just can't be running when you start X.)

Your system may be configured to start X11 as part of the login procedure. If so, skip the rest of this section and the next and start reading at "What to Expect When X Starts."

If your system is not configured to start X11 at login, log into the system in the usual way and type the following command at the command prompt:

```
x11start (Return)
```

You should start the X Window System just once. With X11 running, you should *not* execute the `x11start` command again. Starting X11 and then starting it again while it is still running may cause undesirable results.

Note, however, that you can restart the *window manager* and refresh the *screen* at any time.

Command-Line Options for x11start

In most cases, you will find it convenient to establish environment options in configuration files in your home directory. However, if you don't start X11 automatically at login, you can include environment options on the command line after the `x11start` command. The syntax for this is:

`x11start` [*-clientoptions*] -- [*{path}/server*] [*:display*] [*-options*]

Client Options

Client options pass from the `x11start` command line to all clients in the `.x11start` file that have a `$@` parameter. The options replace the parameter. This method is most often used to specify a display other than the usual one on which to display the client. You can, however, use the command-line option to specify a non-default parameter, such as a different background color.

Server Options

Server options are preceded with a double hyphen (--). If the option following the double hyphen begins with a slash (/) or a path and a slash, it starts a server other than the default server. If the option begins with a colon followed by a digit (:#), it specifies the display number (0 is the default display number). Additional options specified after the server or display refer to the specified server or display. See the `XSERVER` page in the reference section for more information on server options.

Examples

The examples below illustrate starting the X Window System in different ways.

`xllstart`	*The usual way to start X.*

`x11start -bg Blue` *Gives clients followed by* `$@` *a blue background.*
`x11start -- /X2` *Starts server X2 rather than the default server.*

Starting X on a Multi-Seat System

A multi-seat system (a system with more than one display, keyboard, and mouse) requires modification of two X11 configuration files, to allow for more than one display seat. These files, `X*screens` and `X*devices` (where `*` is the number of the display), are located in `/usr/lib/X11`. Each seat must have its own `X*screens` and `X*devices` files. If you have a multi-seat system but have not configured it, see your system installation or configuration manual for more information. Also see "Defining Your Display" in chapter 7.

Starting Seat 0

To start X11 on seat 0 (display 0) of a multi-seat system, log in as usual and type:

`x11start` Return

Seat 0 uses the `/usr/lib/X11/X0screens` and `/usr/lib/X11/X0devices` files to configure its output and input devices. These files are supplied with the system, but you must still match them to your hardware configuration.

Starting Seat 1

To start X11 on seat 1 (display 1) of a multi-seat system, log in as usual and type:

`x11start -- :1` Return

Here the `--` signifies starting the default server while the `:1` specifies sending the output to seat 1. Seat 1 uses the `/usr/lib/X11/X1screens` and `/usr/lib/X11/X1devices` files to configure its output and input devices. If your system has a multi-seat configuration, you must create these configuration files using the `X0screens` and `X0devices` files as models.

What to Expect When X Starts

Whether you start the X Window System from the command line or automatically from a login file, `x11start` always executes the same sequence of steps.

1. If necessary, it adds the path to X11 programs (`/usr/bin/X11`) to your `PATH` variable.
2. It looks in your home directory for a `.x11start` command file to read. If it doesn't find one, it reads `usr/lib/X11/sys.x11start` instead.
3. It starts `xinit`, which starts the server and any clients specified in the `.x11start` command file.
4. It looks in your home directory for a `.Xdefaults` configuration file to read. If it doesn't find one, it reads `/usr/lib/X11/sys.Xdefaults` instead.
5. It reads the configuration file named by the `$ENVIRONMENT` variable, `.Xdefaults-`*hostname* if the variable doesn't exist.

You won't notice any effect from issuing the command until the X display server starts.

The Server Creates the Root Window

When `x11start` starts the server (the program that controls the operation of your keyboard, mouse, and display), your screen will turn gray. This means that the screen has now become the **root window,** the backdrop or "desktop" on which the windows and icons of your environment appear. Although you can completely cover the root window with clients, you can never cover a client with the root window. The root window is *always* the backdrop of your window environment; nothing gets behind it.

In the center of the root window is an hourglass. This is the **pointer** and marks the current screen location of the mouse.

A Terminal Window Appears on the Root Window

A short time later the pointer changes to an `X`, and a terminal window appears at the top of your display (if you're using the default `.x11start` file). This window is under the control of a window manager. If you use the OSF/Motif Window Manager (`mwm`), your window has a three-dimensional frame. This

frame contains window manager controls. (The HP Window manager (`hpwm`) also provides a frame; the `uwm` window manager does not. Refer to appendix A for additional information about `hpwm` and `uwm`.)

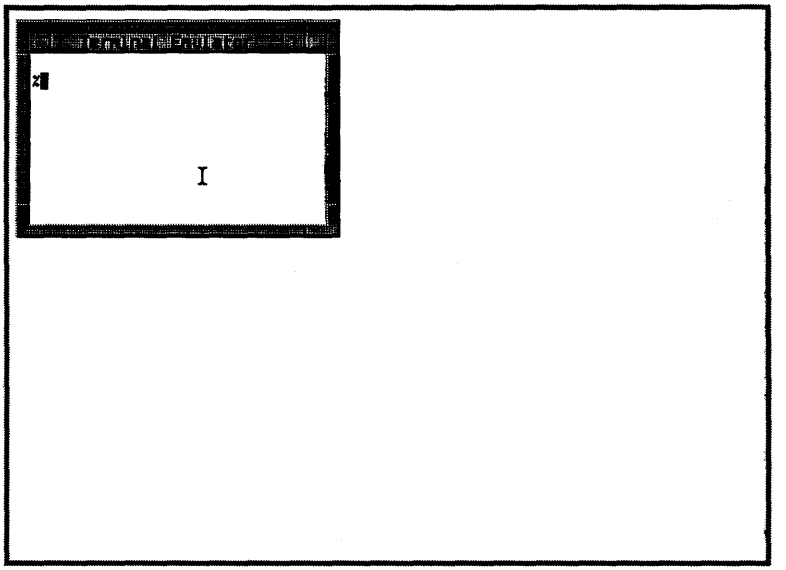


Figure 3-1. The Default X Environment: 'mwm' and One Window

The window that `x11start` creates is an X11 client called `hpterm` and is called an **hpterm window** to distinguish it from other types of window clients. The window contains a command-line prompt and behaves exactly like the screen of a standard HP terminal. You can think of this window as "a terminal in a window."

Move the mouse. The pointer moves on the screen. When the pointer is in the root window, it has an X shape. However, when you move the pointer to a terminal window, the pointer changes to an arrowhead (when on the window frame) or an I (when in the interior of the window).

With the OSF/Motif Window Manager, when you press and release button 1 while the pointer is in a terminal window, the window becomes the **active**

window. When a window is active, its frame changes color. You'll discover that you can't type in a terminal window unless the window is active.

The active window is the terminal window where what you type on the keyboard appears. *Your input always goes to the active window.*

If there is no active window, what you type is lost.

The program running in the active window decides what to do with your typed input. Frequently the program will use a **text cursor** to show where your typed input will be displayed.

What to Do If X11 Doesn't Start

Table 3-1. Possible X Window System Start Problems

If this happens ...	You should do this ...
The message `command not found` appears.	Check your spelling and reenter the start command.
The root window displays for a moment, but then goes blank.	Press the Return key to bring back your original command-line prompt and see below.
The root window displays, but no pointer appears.	Press CTRL Left Shift Reset all at the same time. This brings your original command-line prompt back. See below.
The root window and pointer display, but no terminal window appears.	Press *and hold* button 3. If a menu appears, open a window. Otherwise, press CTRL Shift Reset and try restarting X, then see below.
The terminal window displays, but what you type doesn't appear after the window's command prompt.	Move the pointer into the window and click (press and release) button 1, then type.

If you encounter problems starting X11 for the first time, check the following areas:

- Check the X11 start log in your home directory for clues by typing

```
more .x11startlog (Return)
```

- Check that the correct directory is in your PATH statement. If you do not have an entry for `/usr/bin/X11:.`, `x11start` will add that entry before `/usr/bin:.` in the path. You can be sure that the entry is always there by adding it to the path yourself. To check the PATH variable, type

  ```
  env (Return)
  ```

- Check that the `DISPLAY` environment variable is set correctly. If you do not already have an entry for either `local:0.0` or *host:0.0* (where *host* is the hostname of your system), X11 will add it for you when X11 starts. You can add the entry yourself. To check the `DISPLAY` environment variable, type:

  ```
  env (Return)
  ```

- Check that you have the correct permissions for the `.x11start` file in your home directory. Type:

  ```
  ll .x11start (Return)
  ```

  The resulting permission should be at least:

  ```
  -rwx------
  ```

- Check the `.x11start` file in your home directory for errors. Compare it with the `/usr/lib/X11/sys.x11start` file.

If none of the above seems to help, or you're not sure how to proceed, see your system administrator.

Working With Windows

This section explains features of the OSF/Motif Window Manager (`mwm`). If you have another window manager, some features may work differently from what is described in this manual. Appendix A explains differences between `mwm`, `hpwm`, and `uwm` window managers.

To check which window manager you are using:

1. Move to your home directory by typing

   ```
   cd (Return)
   ```

2. Type:

```
more .X11start (Return)
```

If the system replies "no such file or directory", type:

```
more /usr/lib/sys.X11start (Return)
```

Table 3-2. Which Window Manager Are You Using?

If you see this line ...	You are using this window manager ...
`mwm &`	OSF/Motif Window Manager
`hpwm &`	HP Window Manager
`uwm &`	uwm window manager

In the typical X environment, you have two tools to control window operations:

- The mouse.
- The window manager.

For most window operations, you'll use a combination of the window manager and mouse. (If you lack the space on your desktop, or feel more comfortable with a keyboard, you can configure your keyboard to take the place of the mouse.)

Which Mouse Button Does What

The X Window System works with either a two-button mouse or a three-button mouse. If you have a two-button mouse, you can emulate a three-button mouse. The following table explains which button is which.

Table 3-3. Which Mouse Button Is Which

To press this ...	On a 2-button mouse press ...	On a 3-button mouse press ...
Button 1	the left button	the left button
Button 2	both buttons	the middle button
Button 3	the right button	the right button

Besides using the mouse to point with, you use the mouse buttons to select an operation to be performed on the object pointed to. Buttons have the following actions associated with them:

Press	To hold down a button.
Click	To press *and release* a button without moving the pointer.
Double-click	To click a button twice in rapid succession.
Drag	To press *and hold down* a button while moving the pointer.

The Anatomy of an mwm Window Frame

The OSF/Motif Window Manager surrounds each window on the root window with a functional frame. Positioning the pointer on a part of the frame and performing a mouse button action will execute the function of that part of the frame.

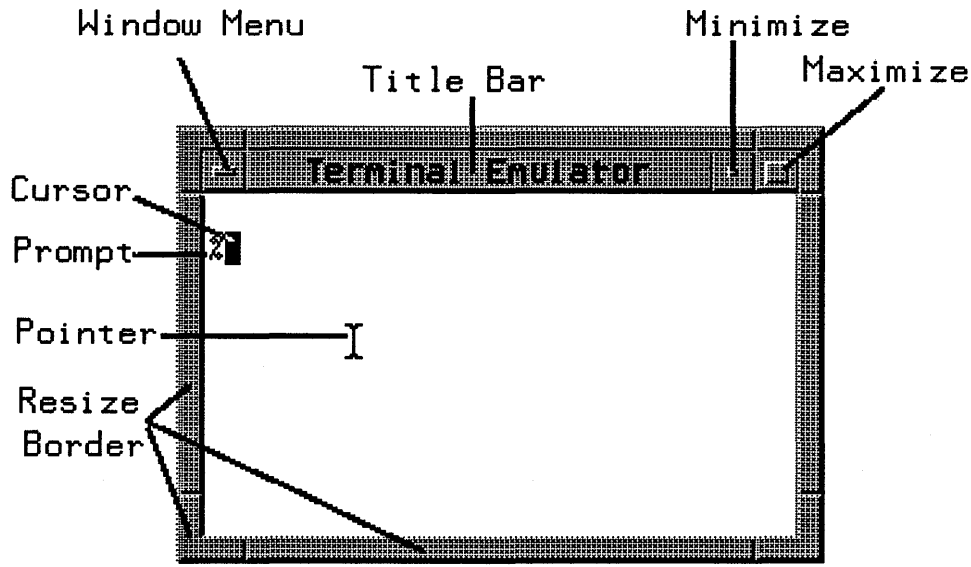


Figure 3-2. The Window Manager Surrounds a Window with a Frame

The parts of the mwm window manager, their functions, and the required mouse operations are listed in the following table.

Table 3-4. Window Frame Parts and What They Do

Frame Part	Function	Mouse Action
Title area	Move a window.	Press and drag button 1.
Window menu button	Display a window menu.	Press button 1.
Window menu button	Select a window menu item.	Press and drag button 1.
Window menu button	Close a window.	Double press button 1.
Minimize button	Iconify a window.	Press button 1.
Maximize button	Expand window to maximum size.	Press button 1.
Frame border	Stretch or shrink a window horizontally, vertically, or diagonally (in two directions).	Press and drag button 1.
Frame and window	Keyboard focus selection.	Press button 1.
Frame and window	(On focus selection) Top window.	Press button 1.

Activating a Window

You make a window active by moving the pointer to any part of the window and clicking button 1 of the mouse. When a window is active, you can interact with it.

Displaying and Selecting from the Window Menu

Every window has a window menu. The window menu button of a window is in the upper left corner of the window frame next to the title bar. You can display the window menu at any time by pressing button 1 with the mouse pointer on the window menu button.

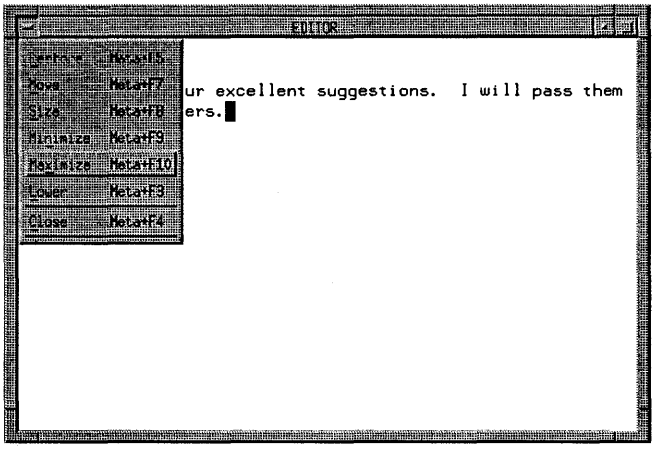


Figure 3-3. Every Window Has a Window Menu

There are three ways to display and use window menus.

Using a Sticky Window Menu

A sticky menu stays displayed until you make a choice. To display the window menu as a sticky menu:

1. Position the pointer on the window menu button.
2. Click button 1.
3. Move the pointer to the selection you want to choose.
4. Click button 1 on that selection. The window menu will disappear and the desired action will take place.

Using a Pulldown Window Menu

To display a window's window menu and make a selection, do the following:

1. Position the pointer on the window menu button.
2. Press *and hold down* button 1.
3. Drag the pointer down the menu to the selection you want to choose.
4. When the selection highlights, release button 1.
5. (Move and Size only.) Move the pointer to the desired location or until the desired size is achieved, then click button 1 to end the operation.

If you change your mind and don't want to make a selection, move the pointer off the menu area *before* you release the button 1.

Using the Keyboard to Display the Window Menu

You can also display the window menu by pressing Left Shift ESC. To make a choice using this method, use the ▲ and ▼ keys to highlight a selection, then press Return. If you don't want to make a selection, press Left Shift ESC again.

Window Manager Selections

The following table describes the window menu selections.

Table 3-5. The Window Menu Selections

To do this ...	Select ...
Restore a window from an icon or after maximizing.	Restore
Change the location of a window.	Move
Change the width and height of a window.	Size
Shrink a window to its icon (graphic representation).	Minimize
Enlarge a window to cover the entire root window.	Maximize
Send a window to the back or bottom of the window stack, the position closest to the root window.	Lower
Immediately stop the window and make it disappear.	Close

You can also use mnemonics and accelerators to select items from the window menu. An accelerator is a key that selects a menu item without posting the menu. For example, the accelerator Alt F9 minimizes a window. The accelerators are shown on the right side of the menu items.

Mnemonics let you select a menu item once the menu has been posted. The mnemonic for a menu item is indicated by an underlined character in its label. To select a menu item using its mnemonic, press the unshifted key for the underlined charcter.

The rest of this chapter explains how you can use the mouse and the window manager to control the windows in your environment.

Moving a Window around the Screen

You can move any window (except the root window) by doing the following:

1. Position the mouse pointer in the title bar.
2. Grab the title bar by pressing *and holding down* button 1.
3. Drag the pointer. An outline of the window shows you the window's new location.
4. Position the outline and release button 1 to relocate the window.

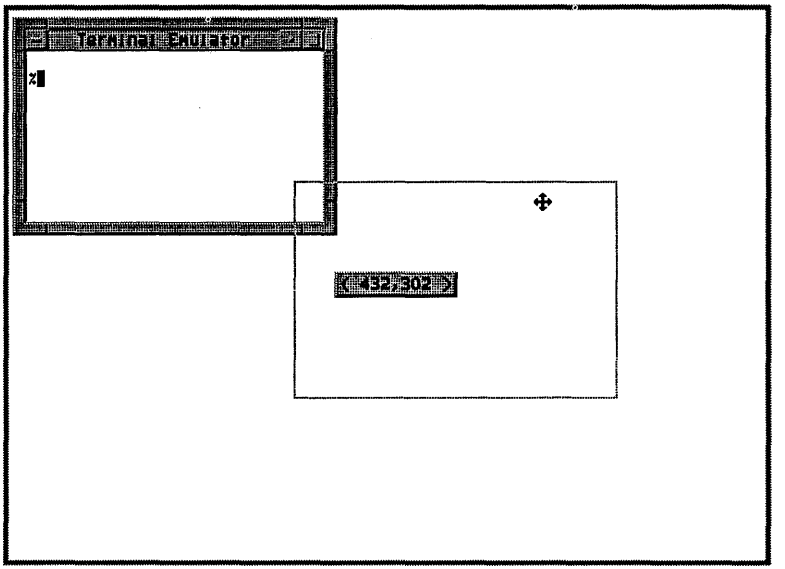


Figure 3-4. An Outline Shows the Window's Location

You will notice that, along with the window outline, a small location box displays at the center of the screen. The numbers in this box are the column and row position of the upper left corner of the actual window (the area inside the window frame). The measurement is in **pixels.** Pixels (short for picture elements) are tiny dots, arranged in rows and columns on the screen, that make up the displayed images.

As mentioned in the previous section, you can also move a window by choosing the "Move" selection from the window menu.

Changing the Size of a Window

To change the size of a window, grab the window's frame with the pointer, drag the frame to the desired size, and then release the frame.

Where you grab the frame determines how the window gets resized. If you grab the side of the frame, the window stretches or shrinks horizontally. If you grab the top or bottom of the frame, the window stretches or shrinks vertically. If

you grab the frame by one of the corner pieces, you can expand or contract the size of the window in two directions at once.

Table 3-6. Where to Grab a Window Frame

If you want to stretch or shrink the window ...	Position the pointer on the ...
vertically from the ...	
top	top of the frame, above the title bar
bottom	bottom of the frame
horizontally from the ...	
right	right side of the frame
left	left side of the frame
diagonally from the ...	
bottom left corner	frame's lower left corner
top left	frame's upper left corner
top right	frame's upper right corner
bottom left	frame's lower right corner

The pointer changes shape when you're positioned correctly for the grab.

Follow these steps to grab and resize the window:

1. Position the mouse pointer on a part of the window frame.
2. Press *and hold* button 1.
3. Drag the mouse pointer. An elastic outline represents the new window size.
4. Release button 1 when the elastic outline is the correct size.

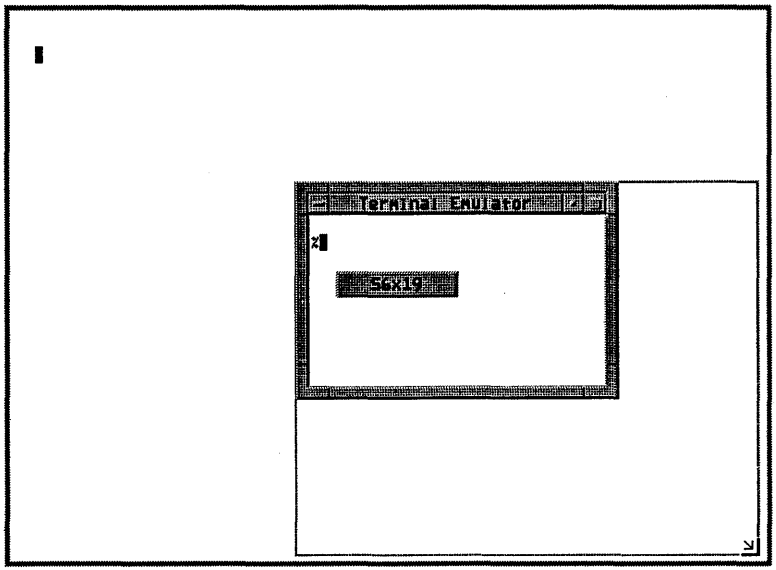


Figure 3-5. An Elastic Outline Shows the Window Size

Although you change a window's size and shape during a resize operation, you do not change its position. The section of the frame opposite where you grab always remains in the same location.

As mentioned earlier, you can also resize a window by choosing the "Size" selection from the window menu. If you choose the "Size" selection, you must cross the window frame's border with the pointer before the elastic outline appears.

Raising a Window to the Top of the Window Stack

As you open more and more windows during a work session, your screen will become cluttered as some windows become obscured under other windows. The windows appear "stacked" on top of one another.

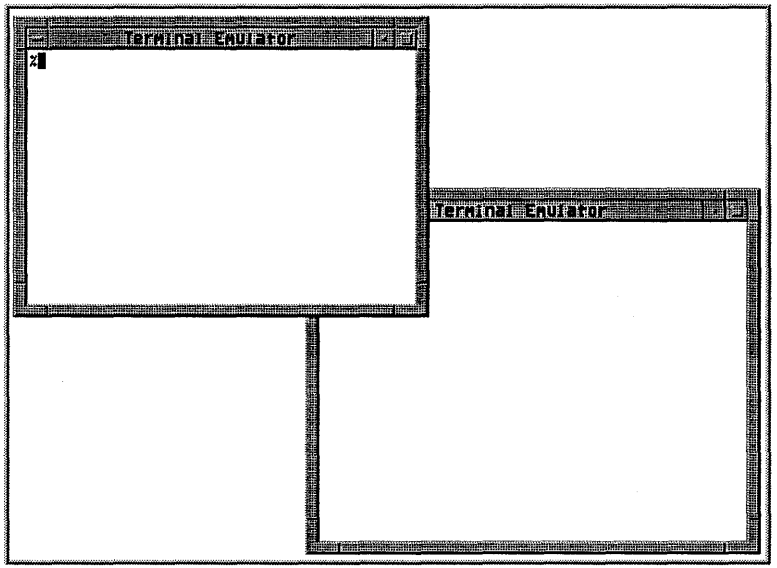


Figure 3-6. Windows Become Obscured by Other Windows

To raise a window to the top of the stack (front of the screen), position the pointer on any visible piece of the obscured window and click button 1. This also makes the window the active window.

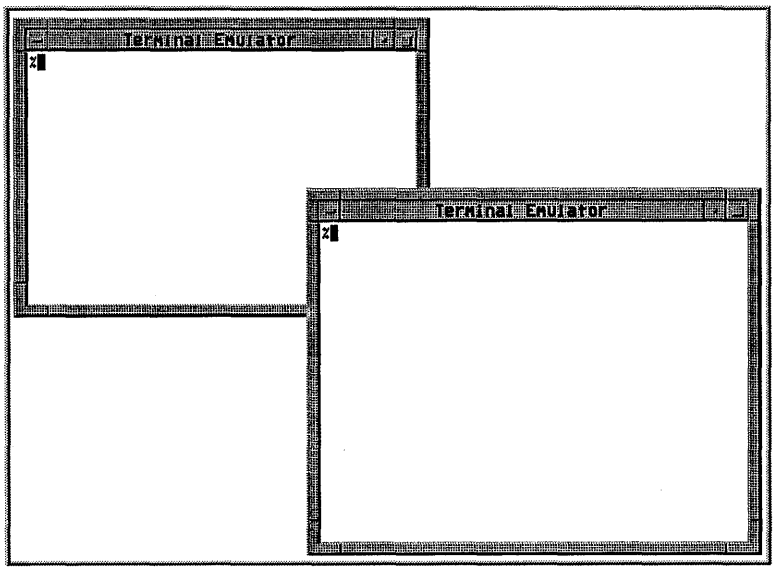


Figure 3-7. A Window Is Unobscured by Raising It

An alternative in some situations is to lower the window on top of the stack by choosing the "Lower" selection from that window menu.

Iconifying a Window

Sometimes raising a window isn't enough to solve the problem of a cluttered root window. You can save space and bring order to your workspace by reducing inactive windows to icons—small, easily-recognizable graphic images that represent full-sized windows. Later, as you need them, you can change the icons back into full-sized windows.

Figure 3-8. Pressing the Minimize Button Iconifies a Window

Changing a window into an icon is known as **iconifying** or **minimizing** the window. To iconify a window:

1. Move the pointer to the minimize button located in the upper right corner of the window frame between the title bar and the Maximize button.
2. Press *and release* button 1.

Immediately after you release button 1, the window is iconified. Successive icons are placed from left to right in a row along the bottom of the root window using a grid pattern. This placement is by default and can be changed if your needs require it.

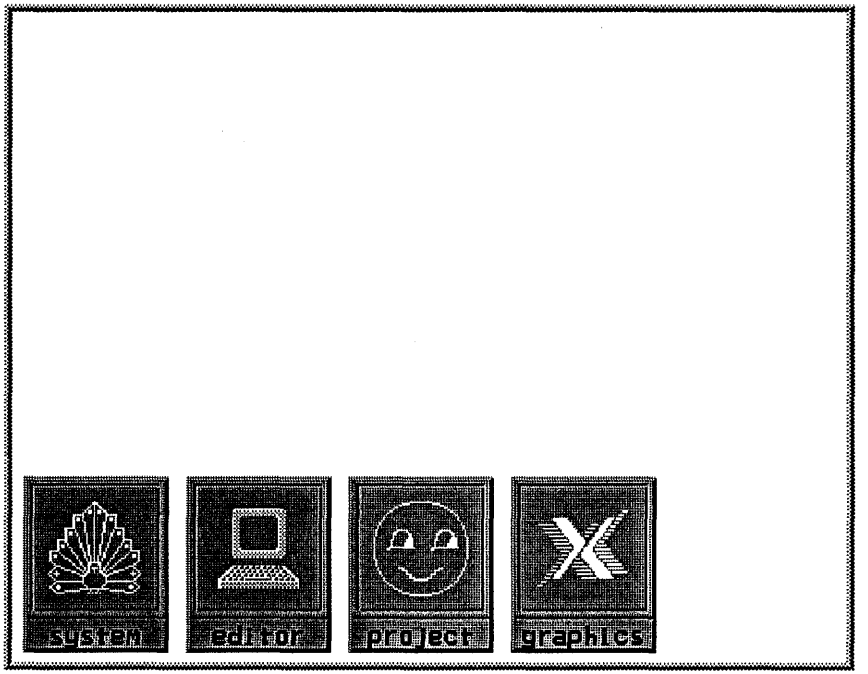


Figure 3-9. Default Icon Placement Is along the Screen's Bottom

You can also change a window into an icon by choosing the "Minimize" selection of the window menu as discussed earlier.

Turning an Icon Back into a Window

When you have room on the root window, or simply want to check the progress of an application running in an iconified window, you can turn the icon back into a window. Changing an icon into a window is called **normalizing** or **restoring.**

1. Move the pointer to the icon.
2. Double-click button 1 (press *and release* it twice in rapid succession).

After you double-click on the icon, the window will reappear located at its previous (pre-iconified) position.

More Work with Icons

Although you can't enter information into an icon, any program running in a window as it is iconified continues uninterrupted until it either completes or pauses to await input from you.

Icons allow you to start an application in a window and then collapse the window into a tiny symbol over in the corner of your screen. There the program quietly does its work without cluttering up your workspace.

Displaying and Selecting from an Icon's Menu

Although an icon doesn't have a frame like a window, it does have a window menu that gives you most standard control options. "Size" and "Minimize" appear on the menu but don't function with iconified windows.

To display an icon's window menu and make a selection:

1. Move the mouse pointer over the icon.
2. Click button 1 to activate the icon and display the menu.
3. Move the mouse pointer to the selection you want.
4. Click button one to make the selection.

To make no selection, move the pointer to the root menu and click button 1. The icon will stay active until you make another window or icon active.

Moving Icons around the Screen

Although icons appear by default in a row along the bottom of the screen, you can move them anywhere on the root window.

To move an icon:

1. Move the mouse pointer onto the icon.
2. Press *and hold* button 1.
3. Drag the pointer to a new location. An outline of the icon shows the current location.
4. Release button 1.

Displaying and Selecting from the Root Menu

The root window has its own menu called (not surprisingly) the **root menu.** You can display the root menu any time the mouse pointer is on the root window. When the pointer is in the root window, remember, it has an × shape.

To display and select from the root menu:

1. Position the pointer anywhere in the root window.
2. Press *and hold* button 3 to display the menu.
3. Drag the pointer down the menu until you have highlighted the desired selection.
4. Release button 3.

To make no selection, move the pointer off the menu *before* you release button 3.

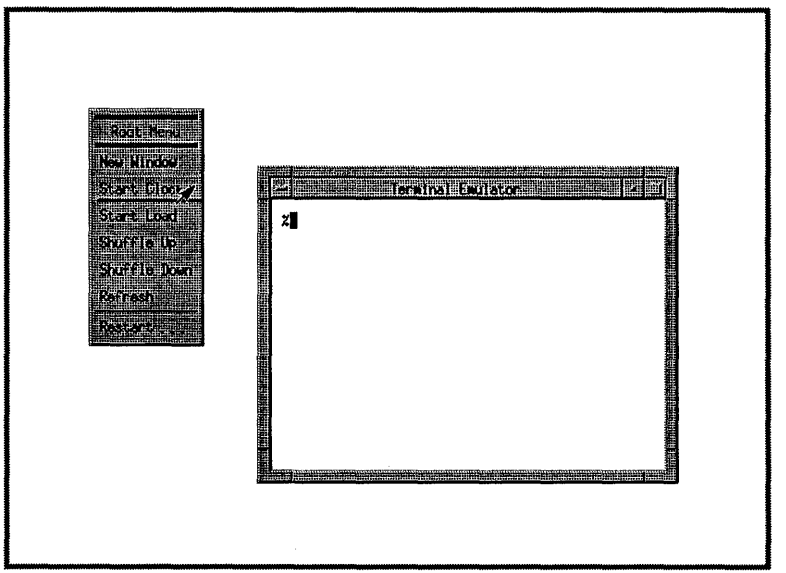


Figure 3-10. The Root Menu Provides Screen-Wide Functions

The default selections of the root menu provide you with screen-wide functions not appropriate for an individual window's window menu.

Table 3-7. What the Root Menu Default Selections Do

To do this ...	Choose this selection ...
Make a new 80×24 `hpterm` terminal window near center screen.	New Window
Display an analog clock in the upper right corner of the root window.	Start Clock
Display a histogram measuring system load (displays next to the clock).	Start Load
Bring the most concealed window to the front of the window stack.	Shuffle Up
Lower the least concealed window to the bottom of the window stack.	Shuffle Down
Blank out then redisplay the screen (useful if video images become corrupt).	Refresh
Restart window manager to see recent configuration changes.	Restart

Exiting From the X Window System

Exiting from the X Window System means stopping the X11 display server. Leaving X places you back at the command prompt you had immediately after you logged into your system.

Before stopping the X Window System, you must first stop any application programs you may have running. This ensures that you do not unknowingly leave any orphaned processes executing. It also ensures that all open files are properly closed to prevent loss of data.

Caution

Stop all application programs before stopping the window system. If you don't do this, any open files may not be updated properly. This could result in the loss of valuable data.

Stopping Application Programs

You can stop a program and remove its window in three ways.

Following the Program's Normal Exit Procedure

The best way to exit a program is to use the program's usual "exit" procedure. This should always be your preferred method for stopping the program. Many programs have commands or keystrokes that stop them.

If the program is a client and created its own window, the window is removed when the client stops. If the program is a non-client in a terminal window, the window remains, and you can stop it when you stop the display server.

Chapter 4 contains more information about stopping programs and what to do if you have trouble stopping a program.

Closing the Window

You can also stop most applications by closing the window in which the application is running. To close a window:

1. Position the pointer on the window menu button.
2. Press *and hold* button 1.
3. Drag the pointer to Close.
4. Release button 1.

Stopping the Window System

After stopping all application programs, stop the window system by holding down the CTRL and Left Shift keys, and then pressing the Reset key. This stops the display server, and with it the window system.

What Next

Now that you've experienced the X Window System and learned how to control your terminal window, you're ready to use X as your working environment.

Chapter 4 contains information about the viewable clients supplied with the X Window System and how to run them from the command-line of a window. Chapter 5 describes how you can incorporate these clients into your environment.

Successive chapters supply increasingly more detailed information about the OSF/Motif Window Manager and other "non-viewable" clients.

4

Running from the Command Line

You can divide the programs you run in your X environment into two groups:

clients Programs written specifically to take advantage of the windowing capability of the X Window System. Clients are the tools you use to work in your X environment.

non-clients Programs written for terminals, not window systems. You can run a non-client in the X Window System by creating a terminal emulation window in which to run the non-client.

This manual uses clients to mean "window-smart" applications, non-clients to mean "terminal-based" applications, and "programs" to refer to both clients and non-clients.

You will probably start the programs that you use frequently either automatically, as part of your X environment, or by choosing them from a menu. However, you can start any client from a command-line prompt.

This chapter discusses the following topics:

- X11 clients and what they do.
- Command-line syntax.
- Starting programs from the command line.
- Stopping programs.
- The `hpterm` terminal emulation client.
- The `xterm` terminal emulation client.
- The `xclock` client.
- The `xload` client.
- Working with common client options.

- Troubleshooting command-line programs.

Meeting the X11 Clients

This chapter discusses four clients (`hpterm`, `xterm`, `xclock`, and `xload`). Other clients are discussed in the following chapters as the functions they control are discussed. But to give you an idea of the tools that are available in the X environment, this section gives you a brief overview of X11 clients and client options.

What the X11 Clients Do

The following tables group the X11 clients (somewhat artificially) into functional categories and give you a brief idea of what the clients do.

Table 4-1. X11 Clients That Initialize and Configure

To do this ...	Use this client ...
Initialize the X Window System and start the X server.	`xinit`
Start `xinit`, X, and X clients.	`x11start`
Alter the modifier-key mappings of a keyboard.	`xmodmap`
Adjust display preference options.	`xset`
Initialize a new colormap for an X environment.	`xinitcolormap`
Create a color database for X.	`rgb`
Add a new remote host to your system.	`xhost`
Load a window manager's resource configuration into the server.	`xrdb`
Compile a BDF-formatted font into an X server format.	`bdftosnf`
Create `fonts.dir` file	`mkfontdir`

Table 4-2. X11 Clients That Control Window Management

To do this ...	Use this client ...
Resize the contents of a window (not the window).	`resize`
Repaint the display screen.	`xrefresh`
Find out information about windows.	`xwininfo`
Provide OSF/Motif Window Manager services.	`mwm`
Provide HP window manager services.	`hpwm`
Provide uwm window manager.	`uwm`

Table 4-3. X11 Clients That Control Graphics Functions

To do this ...	Use this client ...
Open a window into a graphics workstation overlay plane.	`xseethru`
Make a screen dump (pixmap).	`xwd`
Translate an `xwd` pixmap to Starbase format.	`xwd2sb`
Translate a Starbase pixmap to `xwd` format.	`sb2xwd`
Print a screen dump on a PCL-format printer.	`xpr`
Stop multiple Starbase X windows.	`gwindstop`
Create a new X window for Starbase.	`xwcreate`
Destroy a Starbase X window.	`xwdestroy`
Display a previously made screen dump.	`xwud`

Table 4-4. X11 Clients That Provide Viewable Services

To do this ...	Use this client ...
Make a window that emulates an HP terminal.	`hpterm`
Make a window that emulates a DEC or Tektronix terminal.	`xterm`
Display a clock telling the system time.	`xclock`
Display a histogram telling the system load.	`xload`
Make a bitmap for a cursor, icon, or root window tile.	`bitmap`
Display the characters of a particular X font.	`xfd`
Set the color and appearance of the root window.	`xsetroot`

If your interest is in running applications in the X environment, you probably won't ever use some of the clients listed above. If your primary interest is in programming, graphics, or the more technical aspects of environmental control, chapters 6 through 8 and the man pages are your definitive source of information.

The following clients do not require X to be running:

- `rgb`
- `xwd2sb`
- `sb2xwd`
- `xpr`
- `bdftosnf`
- `mkfontdir`

Specifying the General Syntax for Command-Line Starts

Starting clients from the command line of a terminal window gives you a way to dynamically alter the elements that compose your X environment. To start a client from a command line, you must have X11 running, and you must use the correct command-line syntax.

Specifying the Syntax

The general syntax for all clients that you start from a command line is the same:

client [*-options*] [&] (Return)

Options enable you to control the appearance and behavior of a client that you start from a command line. Each client has its own options, but some clients, such as the viewable clients discussed later in this chapter, use the same options. The reference section contains the complete list of all client options.

You specify an option after the client name. The option begins with a hyphen (−) and includes the option itself and an argument. For example, the following is a typical command line to start an `hpterm` window with a black background and white foreground:

```
hpterm -bg Black -fg White & (Return)
```

Choosing Background Processing

An important element of the command-line syntax is the ampersand (&) which ends the command line. As mentioned earlier, the & tells the system to start the client as a background process (a process that doesn't require the total attention of the computer). Background processing enables you to have more than one client running at the same time and frees your keyboard for further use.

Although the & is an optional element, and you can choose to run a client as a foreground process if you desire, you will probably find that in most cases, you will use background processing.

Starting Programs

You can start a client either locally or remotely. A **local client** is a program that is running on your "local" system, the same system that is running your X server. A **remote client** is a program that you view from your local display, but the program actually resides and is running on a system other than yours, a "remote" system.

Starting Local Clients

You can start a local client from the command line any time after you've started X11 and have a window displayed that has a command prompt. To start the client, type the name of that client, followed by any options, then press Return.

It isn't necessary to specify options to run the client; just typing the client name and pressing Return will start the client using a list of option default values. System-wide defaults are contained in the `/usr/lib/X11/sys.Xdefaults` file. Options that override these system-wide defaults are contained in the `.Xdefaults` file in your home directory in `/usr/lib/X11/app-defaults/`*client*, and shell variables (`$DISPLAY`). Command-line options, as you might suspect, override both of these default files.

For example, the following gives you the default clock client: an analog clock updated every 60 seconds:

```
xclock  &  Return
```

You can, however, override these defaults and start a clock client with a digital readout in the lower left corner of the screen.

```
xclock  -digital  -geometry  +1-1  &  Return
```

Starting Local Non-Clients

A non-client normally relies on a terminal instead of a window for displaying its output. To start a non-client program in an X11 window environment, you must first create a terminal emulation window, and then run the non-client in that window.

The following example simply creates an **hpterm** window. Using the command prompt in the window, you can operate most HP-UX system commands (the exception being a command like **update** which affects the entire system, not just the X environment).

```
hpterm &  Return
```

The window opens with its command prompt in the same directory as its **parent window,** the window from which it was started.

At any command-line prompt in any X window, you can start a non-client program simply by typing the start command for that program (usually the program's name) followed by a Return. For example, you could type the following at the command-line prompt:

```
banner windows are great  Return
```

The command prints a banner on the window.

Starting Remote Clients

A remote client is an X11 program running on a computer that is *not the same* computer that the X server is running and displaying on. In other words, the hallmark of a remote client is that the client runs on one computer while the output displays on another.

You can start a remote client from the command line any time after you've started the X Window System and have a window with a command prompt. You can start a client on any **remote host** to which your system has access. A remote host is the computer system that runs the remote client.

Gaining Remote Access

To gain access to a remote host, you must meet all of the following criteria:

- Be on a network with other systems. (This manual uses the *NS-ARPA Services* commands in all examples.)
- Have the internet address and hostname of the remote host in your system's **/etc/hosts** file.
- Have a valid login on the remote host.
- Have the remote host listed in the **/etc/X0.hosts** file.

- Have the remote host listed in a `.rhosts` file in your home directory on your local system. (You may also want to have your local system listed in a `.rhost` file *on the remote host.*)

The first three criteria provide basic network capability to your system. You must have them to use the network whether or not you use the X Window System. The last two criteria provide your local X server with the ability to use the network. The **.rhosts** file lists the systems that have permission to use your username and account to access a system without formally logging in. The **X0.hosts** file contains a list of all X11 hosts known to your X server. The "0" signifies that the file is used by display 0 (similarly, display 1 would use an `X1.hosts` file).

Note

A `.rhost` file allows someone to access your login account *without giving a password.* Depending on your situation, this may pose a threat to the security of your system or the network your system is on. Check with your system administrator and carefully analyze your security needs.

Starting the Client

You have two choices when it comes to running clients on a remote host:

- You can log into the remote host and run a client.
- You can start a client remotely without formally logging in.

In either case, you need to select the display on which you want the output to appear.

Selecting the Display

Just as you need to select a remote host on which to run a client, so too you need to select a display on which the client's output appears. Typically this will be the display attached to your system, but it doesn't have to be.

For example, you could be sitting at your system reviewing lab reports kept on a (remote) lab system when you get the idea to show the reports to Turner at another division. You call to make sure Turner is in, then open a window on Turner's system, display the lab report that interested you, and discuss its

significance with Turner without the delay or trouble of making a physical copy of the report and mailing it.

To help you in selecting a display, viewable clients have a `-display` option that allows you to specify on the command line which system is to receive the output. The syntax for the option is as follows:

`-display` *host*:*display*.*screen*

The *host* specifies the hostname of the system where you want the remote client's output to appear (usually your own system). The *display* is the number of the display where the output is to appear (usually 0 on an HP Series 300 and 0-3 on an Series 800). The *screen* is the number of the physical screen were the output is to appear (usually 0).

Examples of Starting Remote Clients

The following examples illustrate several ways of doing the same thing: starting an `xload` client on remote host `hpcvfaa` and displaying it on the console of your local system `hpcvfbb`.

Example 1: Logging In to a Remote Host the Wrong Way. At the command-line prompt of an existing terminal window, you could type the following:

```
rlogin  hpcvfaa  Return
xload  -display  hpcvfbb:0.0  Return
```

Using this command is a mistake in most cases. Note the `&` is missing from the end of the command line. This command would not return a command prompt to the window until you stopped the `xload` client. Your window would effectively be "frozen."

Example 2: Logging In before Running the Client in Background. At the command-line prompt of an existing window, you could type the following:

```
rlogin  hpcvfaa  Return
xload  -display  hpcvfbb:0.0  &  Return
```

Similar to example 1, these two command lines log you in and then start the xload client, this time as a background process. This leaves your original window free for use, but logged into `hpcvfaa` rather than your local system. The display is again to your system's console.

Example 3: Using a Remote Shell to Start a Client. At the command-line prompt of an existing window, you could type the following:

```
remsh hpcvfaa -n /usr/bin/X11/xload -display hpcvfbb:0.0 & Return
```

Respectively, this command starts a remote shell, on remote host `hpcvfaa`, redirects `remsh` input (necessary in this case), starts the client `xload`, and directs output to system `hpcvfbb`, display 0, screen 0, as a background process.

Note that you wisely used the full path to the `xload` client when starting it. This is a good idea, especially in situations where the remote machine might have two versions of the same client (for example, an X10 and an X11 version of `xload`). The `remsh` command does not allow the $PATH variable.

The benefit of using a remote shell instead of a remote login is that a the local system starts only one process (the client) with a remote shell, while with the remote login the local system starts two processes (the remote login and the client).

Starting Remote Non-Clients

Starting a remote non-client is similar to starting a remote client except that before you start the non-client, you must first start a terminal emulation window in which to run the non-client.

You can always log into the remote host and start a non-client. Using an existing window essentially makes that window a "terminal" of the remote host. Output from the non-client appears in the window. When you exit the non-client and the remote host, the window "returns" to the local system.

Starting a non-client using a remote shell such as `remsh`, however, is sometimes inappropriate. To use a remote shell, you must first create a terminal emulation window in which to run the non-client. If the non-client executes too quickly, you may not see the results, since, once the non-client finishes executing, the emulation window to the remote host closes.

Table 4-5. Choosing a Method of Displaying Remote Processes

If you want the window to ...	Do this ...
Remain after you have finished the initial remote process.	Use an existing window to log in to the host before executing the remote command.
Disappear after you're finished with the remote process.	Execute the command as an option of creating a new window.

Example 1: Logging In to a Remote Host before Running the Non-Client

At the command-line prompt of an existing window, you could type the following:

```
rlogin hpcvfaa (Return)
ll (Return)
```

If you are familiar with networks, you probably recognize this command. It simply logs you in to a remote host, `hpcvfaa`, and then uses the HP-UX `ll` command to list the files in your home directory on that host. Remember, operating system commands, because they are part of HP-UX and not the X Window System, are non-clients.

Example 2: Starting a Window That Starts a Remote Non-Client

This example and the next one show what happens when the same command syntax is used to start different types of remote non-clients. This example shows a non-client that is not interactive.

At the command-line prompt of an existing window, you could type the following and press (Return):

```
hpterm -display hpcvfbb:0.0 -e remsh hpcvfaa -n ll &
```

This example starts another `hpterm` terminal emulation window client. As the first option of that client (`-display`), the ouput is directed to your local display (`hpcvfbb`). As the second option (`-e`), the `hpterm` client executes a remote shell on `hpcvfaa` that connects the window to a remote host (`hpcvfaa`) and lists the files in your home directory there.

Although, at first glance, this command line appears to do the same thing as example 1, there is an important difference. When the `ll` command of example 2 finishes executing, the window created for it to run in will disappear *whether or not you've had time to view all the files.* Remember, the window will close when the remote command has finished executing. Therefore, this is a poor command syntax to use in this situation.

Example 3: Starting a Remote Non-Client Window

At the command-line prompt of an existing window, you could type the following and press Return:

```
hpterm -display hpcvfbb:0.0 -e remsh hpcvfaa -n vi report &
```

This example is the same as example 2 except that the non-client started is different. The non-client `vi` is interactive, that is, you issue commands to it and specifically tell it when you are finished. You start `vi` and open the `report` file. In this case, the window stays displayed until you exit `vi`. You could edit `report` and exit, closing the window. Or you could save `report` and read in another file. As long as you didn't exit `vi`, your "remote editing window" would stay displayed.

Stopping Programs

How you stop a program you've started from a command line depends on whether the program is a client or non-client.

Stopping Clients

Clients like `xload` and `xclock` have no data to save. You stop them by choosing the "Close" selection from the window menu.

Other clients, like `hpterm`, `xterm`, and `bitmap`, may contain data you want to save. Save the data *before* you stop the client. In the case of terminal windows, a non-client running in the window may actually contain the data. Stop the non-client in the approved manner before you stop the window. When you have a command-line prompt in a terminal window, you can stop the window.

In the case of `bitmap`, use the "Write Output" selection on the sidebar menu to save the bitmap before you stop the client.

After you have saved any data and exited any non-clients (in the case of terminal windows), stop the client by choosing the "Close" selection from the client's window menu. Note that if you started a non-client as an option of creating a window, when you stop the non-client, the window will stop.

Stopping Non-Clients

Stop all non-clients in the manner approved in the instructions for that non-client. Generally, a non-client program stops automatically when it finishes executing or has a "stop" provision.

Killing Programs That Won't Stop

If for some reason (and you will no doubt discover some) you cannot stop a program in the normal manner, you should "kill" the program before you exit the window system. Killing the program means using the HP-UX `kill` command to stop the program's execution environment or "process."

Other Ways to Stop a Program

Before you use the `kill` command to stop a program's process, try the following key sequences:

- Press CTRL and, while holding it down, press c.
- Press CTRL and, while holding it down, press d.
- Press q.
- Press ESC, then :, then q.

Killing the Program's Process

If none of these key sequences stop the program, use the following steps to `kill` the program's process:

1. Save any data that needs saving.
2. Find the PID (process ID) for the program by typing the following:

   `ps -fu` *username* Return

where *username* is your login name. The `ps -fu` command lists all the processes running under your login name. You should be able to identify the program you want to kill by looking for it under the "COMMAND" column (the rightmost column in the list). The PID for the program will be located in the second column from the left.

3. To kill the program, type:

   `kill -2` *pid* (Return) *The equivalent of* (CTRL) (c).

   where *pid* is the PID number.

4. If this doesn't work, type:

   `kill -3` *pid* (Return) *A stronger version of* `kill`.

5. If this still doesn't work, type:

   `kill -9` *pid* (Return) *The strongest version of* `kill`.

You can kill several programs at once by including several PIDs separated by spaces in the command. Just be careful that you have the correct PIDs.

Terminal Emulation Clients

The X Window System comes with the following two terminal emulation clients:

`hpterm`	Emulates a Term0 terminal.
`xterm`	Emulates DEC VT102 and Tektronix 4014 terminals.

Emulating an HP Terminal with the 'hpterm' Client

The `hpterm` terminal emulation window is the default terminal used by your X Window System and provides you with basic access to your system. The window's command-line prompt functions exactly like the command-line prompt of an HP **Term0** terminal. Term0 defines an HP level 0 terminal; it is a reference standard defining basic terminal features. For more information about Term0 terminals, see *Term0 Reference* in the HP-UX documentation set.

The `hpterm` window client includes the following features:

- Escape sequences that control terminal operation.
- 16 definable softkeys.
- Full Roman8 character set (ASCII and Roman Extension), ISO 8859.1 character set, and Roman Extension 7-bit characters set.
- Two character fonts (base and alternate).
- Screen editing functions.

If your needs require one or more of these features, see chapter 7, "Customizing Special X Environments," where they are discussed in detail.

Syntax

The syntax of the `hpterm` window client is as follows:

`hpterm` [`-`*options*] [`&`]

You'll find a list of common viewable-client options in "Working with Common Client Options" later in this chapter. For a complete list of `hpterm` options, see the `hpterm` pages in the Reference section.

Using 'hpterm' Terminal Window Softkeys

The `hpterm` client softkeys work exactly like an HP Term0 terminal's softkeys. To display `hpterm` softkeys, position the pointer in an `hpterm` window and press the Menu key. Clicking on a softkey selects that function or setting. Pressing the Menu key again turns off the softkey display.

Additionally, you can color the following elements of `hpterm` softkeys:

- Background.
- Foreground.
- Top shadow.
- Bottom shadow.
- Top shadow tile.
- Bottom shadow tile.

Coloring `hpterm` softkeys is similar to coloring other clients and to coloring the HP Window Manager. You'll find more information about coloring in chapters 5 and 6.

Coloring 'hpterm' Scrollbars

The `hpterm` client also has an option for displaying scrollbars. Scrollbars enable you to scroll the contents of a window, for example, a text file you are editing. You can specify the color and the width for `hpterm` scrollbars. This is also covered in chapter 5.

Emulating a DEC or Tektronix Terminal

The `xterm` client is a terminal emulation window. `xterm` windows emulate DEC VT102 and Tektronix 4014 terminals. Although `xterm` windows are not the default terminal windows for the X Window System, you can use them as your needs require.

Syntax

The syntax of the `xterm` window client is as follows:

`xterm` [`-`*options*] [`&`]

You'll find a list of common viewable client options in "Working with Common Client Options" later in this chapter. For a complete list of `xterm` options, see the `xterm` pages in the reference section.

Using 'xterm' Scroll Features

The `xterm` client has a "jump scroll" option (`-j`). The option enables `xterm`, when its scrolling gets behind, to scroll (jump) several lines at a time from the top of the window.

Another option (`-s`), enables `xterm` to scroll asynchronously. This enables `xterm` to scroll faster when the window screen is no longer up to date because of a high network load.

To use either option, include the option on the command line after the name of the client.

Using 'xterm' Menus

The `xterm` client has three menus. The standard `xterm` menu pops up when the "control" key and button 1 are pressed while the pointer is inside the `xterm` window. The "Modes" menu pops up when the "control" key and button 2 are pressed while the pointer is in the window. The "Tektronix" menu pops up when the "control" key and button 2 are pressed in a Tektronix window.

Special Terminal Emulator Options

Both `hpterm` and `xterm`, because they are terminal emulators, have some special options that other clients don't have.

Making a Login Window

Both `hpterm` and `xterm` have an option that allows you to specify that the window runs a login shell before displaying the command-line prompt. Using the `-ls` option, the shell runs as a login shell, that is, the shell reads the `/etc/csh.login` file and `.login`, or `/etc/profile` and `.profile` file before starting the window.

Cutting and Pasting with the Mouse

Both `hpterm` and `xterm` allow you to use the mouse for cut and paste operations. You can cut text from one location in a window to another, or from one window to another.

Currently, `hpterm` and `xterm` use the button definitions in the following table for cut and paste operations:

Table 4-6.
Mouse Button Definitions for Cut and Paste Operations

If you see ...	On a 2-button mouse press ...	On a 3-button mouse press ...
Button 1	The left button.	The left button.
Button 2	Both buttons simultaneously	The middle button.
Button 3	The right button.	The right button.

To cut and paste using 'hpterm'.

Cutting text

To cut text, follow these steps:

1. Press *and hold* the Shift key.
2. Position the pointer at the start of the text you want to cut and press *and hold* button 2. This marks the beginning of the text region.
3. Drag the pointer to the end of the text you want to cut and release the button. This copies the text into a global **cut buffer,** a buffer that holds text that has been edited out. The region is marked as you drag the pointer.

Pasting text

To paste text from the global cut buffer into a window, follow these steps:

1. Press *and hold* the Shift key.
2. Position the pointer in the window in which you want to paste the text. Because the text will appear like it is being typed at the cursor's location, you may need to position the cursor as well.
3. Click button 3 to "type" the text.

Copying a line

To copy a single line of text from one place and paste it in at the cursor location, follow these steps:

1. Press *and hold* the Shift key.
2. Position the pointer at the start of the text you want to copy.
3. Click button 1 to copy text from the pointer to the end of the line and "type" it at the cursor location in the same window. (Position the pointer in another window and click button 3 to "type" the text in the second window.)

To cut and paste using 'xterm'.

Cutting text

To cut text, follow these steps:

1. Position the pointer at the start of the text you want to cut.
2. You can cut a text region in the following three ways:

- To cut a region character by character, click *and hold* button 1.
- To cut a region word by word, double-click *and hold* button 1.
- To cut a region line by line, triple-click *and hold* button 1.

This marks the beginning of the text region.

3. Drag the pointer to the end of the text you want to cut and release the button. This copies the text into the global cut buffer.

Pasting text

To paste text from the global cut buffer into a window, follow these steps:

1. Position the pointer in the window in which you want to paste the text. Because the text will appear like it is being typed at the cursor's location, you may need to position the cursor as well.
2. Click button 2 to "type" the text.

Extending text

You can extend or contract either half of the current selection by following these steps:

1. Position the pointer in the top or bottom half of the text that you have selected with button 1.
2. Press *and hold* button 3.
3. To expand or contract the half that you picked, drag the pointer away from or toward the center point of the selection.
4. When the selected area includes the correct text, release button 3.

Scrollbars

You can start either an `hpterm` or `xterm` window with scrollbars. To do this, include the `-sb` option on the command line when you start the window. For example, to start an `hpterm` window with a scrollbar, type the following line after the command prompt:

```
hpterm -sb & Return
```

Window Titles and Icon Names

By default the title of a terminal emulation window is `Terminal Emulator`. Equally original are the default names that appear on labels of `hpterm` and `xterm` icons. These are, respectively "hpterm" and "xterm." Two options enable you to give your terminal windows and icons more original names if you so desire.

Use the `-title` option to give a title to a terminal emulation window. Titles with two or more words must be enclosed in quotes (*"title1 title2"*).

Use the `-n` option to give a name to the icon of a terminal emulation window. Icon names of two or more words must be enclosed in quotes (*"name1 name2"*). Note also that lengthy names may be truncated on the right to the width of the label.

The following example illustrates the use of these two options:

```
hpterm -n System -title "System Window" & Return
```

This example creates an `hpterm` window, giving it the title "System Window." When the window is iconified, the icon label reads "System."

Telling Times with 'xclock'

The X Window System includes a clock client called `xclock`. You can choose either an analog clock (a clock with hands and a face) or a digital clock (a clock with a text readout showing the day, date, time, and year).

Syntax

The syntax for `xclock` is as follows:

```
xclock [-options] [&]
```

You'll find a list of options that `xclock` shares with other viewable clients in "Working with Common Client Options" later in this chapter. For a complete list of `xclock` options, see the `xclock` pages in the reference section.

Although ampersand (&) is an option, you will rarely find it practical to use `xclock` with out it. When run from the command line as a foreground process (without the `&`), `xclock` takes control of the window and *does not* return the command-line prompt, thus making it impossible for you to use the window until you either close the clock or kill its process.

Some 'xclock' Options

The `xclock` client comes with some options that are unique.

Marking the Half Hours

The `-chime` option causes the speaker on your system to sound once on the half hour and twice on the hour.

Selecting the Clock Format

As mentioned, `xclock` has two formats: analog and digital. The analog format is the default.

Specifying the `-analog` format (or no format) draws a conventional 12-hour clock face with strokes marking the hours and ticks marking the minutes.

Specifying the `-digital` format draws a digital readout containing the day, date, time, and year. The format automatically varies for local language custom based on the value of the $LANG environment variable. (Of course, you must specify an appropriate font for the language you select.) For more information about $LANG, refer to the *HP-UX Native Language Support* manual.

Updating the Time

The `-update` *seconds* option enables you to select the time interval between updates to the clock display. The default is an update every 60 seconds.

Examples

The following examples illustrate both clock formats:

```
xclock -digital -update 10 & Return
```

```
xclock -analog -chime -update 5 & Return
```

The first example creates a digital clock that updates every 10 seconds. The second example creates an analog clock that chimes every 30 minutes and updates every 5 seconds.

Viewing System Load with 'xload'

The X Window System includes a client called `xload` that displays a histogram of the current system load.

Syntax and Options

The syntax for `xload` is as follows:

`xload` [`-`*options*] [`&`]

You'll find a list of options that `xload` shares with other viewable clients in "Working with Common Client Options" later in this chapter. For a complete list of `xload` options, see the `xload` pages in the reference section.

As with `xclock`, the `&` that completes an `xload` command line is, strictly speaking, an option. But you will rarely find it practical to use `xload` without it. When run from the command line as a foreground process (without the `&`), `xload` *does not* return the command-line prompt, thus making it impossible for you to use the window until you either close or kill the `xload` client.

Some 'xload' Options

The `xload` client comes with some options that are unique.

Updating the Load

The `-update` *seconds* option enables you to select the time interval between updates to the load histogram display. The default is an update every 5 seconds.

Scaling the Histogram Graph

The `-scale` *division* option enables you to adjust the scale of the histogram by drawing extra division lines on the graph. By default `xload` measures the average load on the system using a scale of 0 (no load) to 1 (a single division). Using the `-scale` option, however, you can select a division other than 1 against which to measure the load.

Note that if you use the default setting and the system load goes beyond that, extra divisions will be drawn automatically to keep the load in scale.

Example

The following example illustrates an `xload` client started from the command line:

```
xload -update 15 -scale 2 & Return
```

This example creates a load histogram that updates every 15 seconds and uses a scale of 2 units.

Working with Common Client Options

The viewable clients have the following options in common:

- Color.
- Display.
- Size and location.
- Fonts.
- Other options.

Color Options

All viewable clients have elements that you can color. If your system uses a monochrome monitor, it is still possible to use the tiling capability of the HP Window Manager to achieve a pleasing 3-D gray-scale color scheme.

The viewable X11 clients, as you might expect, have options for specifying the color of their elements.

Available Client Color Options

The following table lists the colorable elements of X11 clients.

Table 4-7. Color Options for Viewable X11 Clients

Option Descriptions		X11 Clients			
To change this ...	**Use this option ...**	**hpterm**	**xterm**	**xclock**	**xload**
Foreground color.	`-fg` *color*	√	√	√	√
Background color.	`-bg` *color*	√	√	√	√
Cursor color.	`-cr` *color*	√	√		
Pointer color.	`-ms` *color*	√	√		
Clock hands color.	`-hd` *color*			√	
Hand edge color.	`-hl` *color*			√	√

You can specify an element color on the command line in the following two ways:

- By listing the color name after the option.
- By listing the hexadecimal color value after the option.

The file `/usr/lib/X11/rgb.txt` lists all colors that have "names." Specifying a name after a color option causes the element referred to by the option to display in that color.

For example, the following command line creates an `hpterm` window with a black background and a white foreground:

```
hpterm -bg Black -fg White & Return
```

Using Hexadecimal Color Values on the Command Line

While using color names is an easy way to select colors, you are limited by the number of available names. Fortunately, the use of hexadecimal color values offers a solution. You can specify *any* color, whether it has a name or not, by using a hexadecimal color value. This value corresponds to the amount of each of the primary colors (red, green, and blue) that are used to make up the color.

If you use the C shell (`csh`), a color value consists of a number sign (`#`) followed by a hexadecimal number for the value of each color (red, green, blue). If you use the Bourne shell (`sh`) or Korn shell (`ksh`), a color value consists of `\#` followed by a hexadecimal number for each color (red, green, blue). The hexadecimal number can be 1, 2, 3, or 4 digits long. You must have the same number of digits *for each* of the primary colors. Thus, valid color values consist of 3, 6, 9, or 12 hexadecimal digits.

For example, #3a3 and #300a00300 are both valid color values for the same color, a shade of green. #000, #000000, #000000000, and #000000000000 all specify the color black. And #fff, #ffffff, #fffffffff, and #ffffffffffff all specify white. The number of digits you use in color values depends on your need for subtle shades of color and the capability of your display hardware.

Examples

As an example of specifying color on a command line, suppose you wanted an analog clock with a plum background, white foreground, and black hands with white edges. You could specify the clock in either of the two following ways:

```
xclock -bg plum -fg white -hd black -hl white & Return
```

or

```
xclock -bg #c5489b -fg #fff -hd #000 -hl #fff & Return
```

For the purposes of this example, plum, white, and black were chosen because they are colors with valid color names in `/usr/lib/X11/rgb.txt`. However, you can specify a unique color (one with no name equivalent). For example, a slightly darker plum for the background is created with the following hexadecimal value:

```
xclock -bg #ba408b -fg white -hd black -hl white & Return
```

Specifying Size and Location on the Command Line

Each client you add to your environment is located at a certain position on the root window. The default position is the upper left corner, but you can place a client anywhere on the root window using the `-geometry` option.

The Syntax of the '-geometry' Option

The `-geometry` option has the following syntax:

`-geometry` *Width*×*Height*[±*column*±*row*]

Width	The width of the window in characters (for terminal windows) or pixels (for other clients). Note that the width of the terminal window that appears varies depending on the font size.
Height	The height of the window in lines (for terminal windows) or pixels (for other clients). The height of a terminal window is also dependent on the size of the font chosen.
column	The column location of the window given in pixels. Plus (+) values refer to the left side of the window. Minus (−) values refer to the right side of the window.
row	The row location of the window given in pixels. Plus (+) values refer to the top of the window. Minus (−) values refer to the bottom of the window.

You have the following choices for defining client size and location:

- Including both the size and location in the command. The window appears as specified.
- Including only the size in the command. The window appears in the specified size at the default location.
- Including only the location in the command. The window appears at the specified location in its default size.
- Including neither size nor location in the command. The window appears in the default size at the default location.

Placing Clients on the Root Window

The following table lists some typical locations for a 1280×1024 high-resolution display.

Table 4-8.
Example Locations for an 80×24 X11 Terminal Window

To position a window here ...	Use this location ...
The upper left corner of the root window.	+1+1
The lower left corner of the root window.	+1−1
The upper right corner of the root window.	−1+1
The lower right corner of the root window.	−1−1
The left side at mid-window.	+1+512
The right side at mid-window.	−1+512
The top of the root window and right of center.	+635+1
Centered at left.	+1+330
Centered at right.	−1+330
Centered in the root window.	+320+330

Note

The resolution of screens varies. Some locations may work for you but be off the screen for someone else! Therefore, you may need to experiment, altering the geometry specifications to fit the resolution of the screen.

Example

The following examples illustrate a typical command-line use of the geometry option:

```
xclock -geometry 90x90-1-30 & Return

xload -geometry 120x90+1-1 & Return
```

The first example starts an `xclock` client. The geometry option gives the clock a 90-pixel by 90-pixel size and locates it 1 pixel to the left and 30 pixels up from the lower right corner of the screen.

The second example starts an `xload` client. The geometry option gives the client a 120-pixel by 90-pixel size and locates it in the lower left corner of the screen.

Specifying the Display on the Command Line

As described above in "Starting Remote Clients," you can start an X client program on one computer and have the output of the program display on another. The default display is obtained from the `DISPLAY` environment variable of the system on which the client starts, but the `DISPLAY` variable can be reset dynamically for a client by including a `-display` option on the command line when you start the client.

The Syntax for the '-display' Option

The `-display` option has the following syntax:

`-display` [*host:display.screen*]

host	Specifies the hostname of a valid system on the network. Depending on the situation, this could be your system's hostname or the hostname of a remote system.
display	Specifies the number of the display on the system on which you want the output to appear. On HP 9000 S300's, this number will usually be 0. On HP 9000 S800's, this could be any number depending on the configuration.
screen	Specifies the number of the physical CRT screen where the output is to appear. The default is 0.

Example

An example of using the display option on the command line is the following:

```
hpterm  -display  hpcvfaa:0.0 &  Return
```

This command, when issued at a command-line prompt, starts an `hpterm` process on the local system and displays output (the window) on screen 0, display 0 of the `hpcvfaa` system. The window has the default size, location, and color.

Specifying the Font in the Command Line

In addition to the options discussed above, the viewable clients also have an option that enables you to specify the font for text. The `-fn` option enables you to select a font for the label that displays on the `xload` client as well as the text for terminal emulation windows.

Working with Fonts

Fonts are in subdirectories within the `/usr/lib/X11/fonts` directory. You may specify a font to use in either of two ways:

- Providing an "alias" for the font.
- Specifying a list of the font's characteristics.

These methods of specifying fonts are discussed in more detail in chapter 5.

If no font is specified, or if the server can't find the font, `fixed` is usually used.

The two terminal emulators also have a **-fb** option. You can use this option to specify a a font for bold text. The text specified must be the same height and width as the font specified with **-fn**, the "normal" font.

Example

The following examples illustrate the command-line use of the font option:

```
hpterm -fn hp8.10x20 & Return

hpterm -fn hp7.10x20 & Return
```

The first line creates an **hpterm** window with a large, easy-to-read font (hp8.10x20). The font is located in **/usr/lib/X11/fonts/misc/hp8.10x20**, and is referred to by the alias **hp8.10x20**. The second line represents a misspelling of the first line. The result is the creation of a window, but the font used for the command-line prompt is the default font, not hp8.10x20.

For information about fonts, refer to chapter 5.

Where to Go Next

If your X Window System environment meets your present needs, you can stop here. If, however, you would like to customize your environment a little, perhaps coordinate the colors of your clients, or select different clients to display when you start X, or arrange them more efficiently on your root window, you should continue to chapter 5.

Chapter 6 explains, in more detail than the average mortal need be concerned about, how to work with the OSF/Motif Window Manager and its resources to fine-tune your control over your X environment. Chapters 7, 8, and 9 present cases where customization is needed because of special hardware considerations or the extensive use of graphics.

5

Customizing Your Local X Environment

As you become familiar with the X Window System, you will probably want to modify your X environment to better suit your situation. Chapter 5 discusses customizing your window environment. Using the information in this chapter, you can change the appearance and behavior of the X Window System to suit your needs *without affecting the appearance and behavior for other users.* These changes include the following:

- Customizing the colors of clients.
- Changing the clients that start when you start X.
- Starting X at login.
- Creating custom bitmaps.
- Customizing the root window.
- Working with fonts.
- Using Remote Hosts.

Before You Begin Customizing

To customize your window environment, you must modify or create three configuration files. These files contain information that the X server uses to configure your window environment. Incorrectly modifying these files could bring your X Window System to a screeching halt. So if you are new to this type of thing, read the following two sections. They list some simple safety precautions (often overlooked by people who "know what they're doing") that keep you from getting into trouble if you make a mistake. They also give you a little background on the configuration files with which you'll be working.

How to Begin Customizing

Swimming pools you should jump into with both feet; customizing your environment you should approach step by step. Although the following safety tips may take a little more time to implement, they are the steps that people regretfully "wish they had taken" after something has gone wrong.

Making Backup Copies of Your Work

Don't modify any original files. Make a copy of the original file and then modify the copy. That way, if all else fails (and it sometimes does), you can go back and get another copy of the original and start again. As you get deeper into rearranging your environment, test your modifications and, if they work properly, save that version of your modifications, make a copy of it, and continue the rest of your modifications on the copy.

Making Incremental Changes

Make incremental changes when you edit the configuration files. That way, if something goes wrong, you can easily isolate the mistake. It's much easier to pinpoint a mistake in syntax or spelling if you've only modified one line of one file, rather than multiple lines in several files.

Choosing a Text Editor

The three configuration files are text files. You can use `vi`, `emacs`, or any other editor that produces text files to do your editing. You edit the text of the configuration files just like you would edit the text of a letter, replacing what you don't want with something more appropriate.

One trick that you might consider is to comment out a line that you don't want rather than deleting it from the file. To comment out a line, place a number sign or pound sign (`#`) in the left margin of the line (use a `!` to comment out a line in `.Xdefaults`). This allows you to use the line as a model for future editing and provides you with the opportunity to restore it (by uncommenting it) at some future time.

Where to Begin Customizing

Three configuration files come with the X Window System:

- `sys.Xdefaults`
- `sys.x11start`
- `system.mwmrc`

You'll find these files in the `/usr/lib/X11` directory. The files supply system-wide default configuration for users who start X but don't have individual configuration files in their home directories.

The following three configuration files should be in your home directory if you want to customize your X environment. Typically, you copy them from their system-wide versions in `/usr/lib/X11`:

`.Xdefaults`	Specifies default appearance and behavior characteristics for clients.
`.x11start`	Specifies the clients that start when the X Window System starts.
`.mwmrc`	Specifies the menus, menu selections, and button and keyboard bindings that control the OSF/Motif Window Manager. This file is discussed in more detail in chapter 6.

Note that the `/usr/lib/X11/app-defaults/` directory may also contain configuration files for client applications.

Customizing the Colors of Clients

You control the color of the clients (including the window manager) that display in your X environment by modifying the `.Xdefaults` file. Valid color names are stored in `/usr/lib/X11/rgb.txt`.

Coloring window manager features such as borders is covered in "Managing the General Appearance of Window Frames" in chapter 6.

Copying 'sys.Xdefaults' to '.Xdefaults'

When you issue the `x11start` command to start the X Window System, the command looks in your home directory for a `.Xdefaults` file. If it finds the file, it uses the information in the file to color your X environment. If it doesn't find the file, the `x11start` command uses `/usr/lib/X11/sys.Xdefaults`.

To begin customizing the colors of your X environment, copy the `sys.Xdefaults` file to your home directory as `.Xdefaults`.

```
cp  /usr/lib/X11/sys.Xdefaults  $HOME/.Xdefaults  Return
```

This gives you a read-only copy of `.Xdefaults`. You must make the `.Xdefaults` file writable so that you can modify it. To do this, type the following command:

```
chmod  u+w  .Xdefaults  Return
```

This will enable you to color the clients in your environment without affecting the environments of other users on the system.

If the file becomes corrupted and inoperable during the editing process, you can always make a fresh copy from `/usr/lib/X11/sys.Xdefaults` and begin the editing process again.

Changing Client Colors

Changing the color of a particular client element is a simple process. You specify a value for the resource that controls the element you want to color. Use the following steps:

1. Start your text editor and open the `.Xdefaults` file.
2. Scroll down or search for the *client*resource* you want to color.
3. Delete the `!` and the space from the left margin to activate the line.
4. Replace the "`<color>`" at the end of the line with the color you desire.
5. Save the file and exit the text editor.

To view the effect of a change to `.Xdefaults`, simply start a client of the type whose color you modified.

Determining Which Elements to Color

The following tables list the colorable elements of your X environment by client.

Table 5-1. Terminal Window Elements

To color this element ...	Look for this resource ...
`hpterm` window text	! HPterm*foreground:
`hpterm` window background	! HPterm*background:
`hpterm` window text cursor	! HPterm*cursorColor:
`hpterm` window mouse pointer	! HPterm*pointerColor:
`xterm` window text	! XTerm*foreground:
`xterm` window background	! XTerm*background:
`xterm` window text cursor	! XTerm*cursorColor:
`xterm` window mouse pointer	! XTerm*pointerColor:

Table 5-2. Load Histogram Elements

To color this element ...	Look for this resource ...
system load histogram foreground	! Xload*foreground:
system load histogram background	! Xload*background:

Table 5-3. Clock Elements

To color this element ...	Look for this resource ...
analog clock tick marks	! XClock*foreground:
digital clock text	! XClock*foreground:
clock background	! XClock*background:
clock hands	! XClock*hands:
edges of clock hands	! XClock*highlight:

Syntax

At some point, you may want to change the color of an element that is not in your `.Xdefaults` file. You can add that element to the file by typing it in `.Xdefaults` on a line by itself using the following syntax:

*client*resource*: { *color name* / *#hexadecimal* }

For *client,* you can use any valid viewable X client. For *resource,* you can use any valid color resource for that client. The surrounding lines in the file provide you with examples to model your line after. You can find a complete list of the resources for each client in the reference section.

The color you specify can be a color name from the `/usr/lib/X11/rgb.txt` file or a hexadecimal value. While color names are easier to remember, hexadecimal values enable you to specify a greater variety of colors.

A hexadecimal value is composed of three segments, one segment for each of the primary colors red, green, and blue. A hexadecimal value consists of a number sign (`#`), signaling the start of a hexadecimal number, followed by 1, 2, 3, or 4 hexadecimal digits *for each* primary color. Thus a valid color value can be 3, 6, 9, or 12 hexadecimal digits.

For example, #3a3 and #300a00300 are both valid color values for the same color, a shade of green. The number of digits you use in color values depends on your need for subtle shades of color and the capability of your display hardware.

Examples

The following examples illustrate some typical lines in your `.Xdefaults` that color client elements.

```
XClock*foreground:   Black
XClock*background:   White
XClock*hands:        SkyBlue
XClock*highlight:    Black
```

The above lines color the elements of the `xclock` client. The first line makes the tick marks of an analog clock (and the readout of a digital clock) black.

The next line gives it a white background (face). The next two lines color the hands of an analog clock skyblue with black borders.

When coloring client elements, you should usually color adjacent elements in contrasting colors. The obvious mistake is coloring clock hands the same color as the background. Sure, the hands display in the color you select, but it's frightfully hard to tell the time. The same holds true for foregrounds and backgrounds that lack sufficient contrast.

What Colors Are Available

You can color your X11 environment by specifying any of the color names listed in the following table. Type the color name exactly as it appears below.

Table 5-4. X Window System Color Name Table

Available Colors			
Aquamarine	Black	Blue	BlueViolet
Brown	CadetBlue	Coral	CornflowerBlue
Cyan	DarkGreen	DarkOliveGreen	DarkOrchid
DarkSlateBlue	DarkSlateGray	DarkSlateGrey	DarkTurquoise
DimGray	DimGrey	Firebrick	ForestGreen
Gold	Goldenrod	Gray	Green
GreenYellow	Grey	IndianRed	Khaki
LightBlue	LightGray	LightGrey	LightSteelBlue
LimeGreen	Magenta	Maroon	MediumAquamarine
MediumBlue	MediumForestGreen	MediumGoldenrod	MediumOrchid
MediumSeaGreen	MediumSlateBlue	MediumTurquoise	MediumVioletRed
MidnightBlue	Navy	NavyBlue	Orange
OrangeRed	Orchid	PaleGreen	Pink
Plum	Red	Salmon	SeaGreen
Sienna	SkyBlue	SlateBlue	SpringGreen
SteelBlue	Tan	Thistle	Transparent
Turquoise	Violet	VioletRed	Wheat
White	Yellow	YellowGreen	

Where to Find the Available Color Names

All of the color names available in the X Window System are listed in the `/usr/lib/X11/rgb.txt` file. You can find the names of colors by typing the following command to view the file:

```
more /usr/lib/X11/rgb.txt Return
```

The file is several "pages" long, so you may find it more convenient to make a printed copy of the file using the following command:

```
pr -160 -h "X11 Color Table" /usr/lib/X11/rgb.txt | lp Return
```

Determining Where to Color Your Environment

The usual place to specify colors is in the `.Xdefaults` file in your home directory. However, you can change the color of a particular instance of an element (such as the foreground color of a single window) by specifying that color on the command line that starts the client. If you start the client when you start X11, the command line would be in the `.x11start` file. If you start the client from a menu, the command line would be in the `.mwmrc` file.

For example, if you wanted an `hpterm` window to have a DarkSlateGrey background and White foreground, you could specify these colors on the command line you used to start the window.

Coloring a Single Instance of a Client

The following command, issued at the command line prompt, overrides any background and foreground colors specified in the `.Xdefaults` file and creates a single `hpterm` window with a DarkSlateGray background and White foreground.

```
hpterm -bg DarkSlateGrey -fg White & Return
```

This syntax should be familiar to you if you have read chapter 4.

Coloring Windows that Start Automatically

The following line in your `.x11start` file overrides any background and foreground colors specified in the `.Xdefaults` file and creates an `hpterm` window with a DarkSlateGrey background and White foreground *each time* you start X11.

```
hpterm -bg DarkSlateGrey -fg White &
```

Note that the syntax of the above example is exactly like the syntax used when you start a client from the command line.

Coloring Windows that Start from Menus

The following line in your `.mwmrc` file overrides any background and foreground colors specified in the `.Xdefaults` file and, when you choose the `Dark Window` selection from the menu, creates an `hpterm` window with a DarkSlateGrey background and White foreground.

```
"Dark Window" f.exec "hpterm -bg DarkSlateGrey -fg White &"
```

This syntax is similar to the command-line syntax with which you are already familiar. You'll learn more about it in "Managing Window Manager Menus" in chapter 6.

Coloring 'hpterm' Softkeys and Scrollbars

To color `hpterm` softkeys or scrollbars, you may need to add one or more lines from the following table to your `.Xdefaults` file:

Table 5-5.
You Can Color These 'hpterm' Softkey and Scrollbar Elements

To color this element ...	Add this line ...
softkey text	HPterm*softkey*foreground:
softkey background	HPterm*softkey*background:
top and left softkey bevel	HPterm*softkey*topShadowColor:
bottom and right softkey bevel	HPterm*softkey*bottomShadowColor:
top and left softkey bevel tile	HPterm*softkey*topShadowTile:
bottom and right softkey bevel tile	HPterm*softkey*bottomShadowTile:
scrollbar foreground	HPterm*scrollBar*foreground:
scrollbar background	HPterm*scrollBar*background:

The lines you add to the `.Xdefaults` file all have the following syntax:

$$\texttt{HPterm*}\left\{\begin{array}{l}\texttt{softkey}\\ \texttt{scrollBar}\end{array}\right\}\texttt{*}\textit{resource}\texttt{:}\ \left\{\begin{array}{l}\textit{color}\\ \textit{\#hexadecimal}\end{array}\right\}$$

The color you select can be either a color name (Magenta) from the `rgb.txt` file or a hexadecimal value (#ffe000ffe).

For tile, you can select a number of tile "patterns." For a complete list see "Changing the Tiling of Window Frames" in Chapter 6.

Changing the Clients that Start When You Start X

By modifying the `.x11start` file in your home directory, you can control which clients appear as part of your environment when you start X.

Copying 'sys.x11start' to '.x11start'

The clients that start by default when you start X are specified by command lines in the `sys.x11start` file. To change the clients that start in your personal X environment from the default (`hpwm` and an `hpterm` window), copy `sys.x11start` from the `/usr/lib/X11` directory to your home directory.

```
cp  /usr/lib/X11/sys.x11start  $HOME/.x11start  Return
```

This gives you a read-only copy of `.x11start`. You must make the `.x11start` file writable so that you can modify it. To do this type, the following command:

```
chmod  u+w  .x11start  Return
```

This will enable you to change the clients that start in your environment without affecting the environments of other users on the system.

If you accidentally ruin the `.x11start` file during the editing process, you can always make a fresh copy from `/usr/lib/X11/sys.x11start` and begin the editing process again.

Viewing X11 Start Error Messages

The **x11start** command records any messages that occur as X11 starts. Viewing these messages is an important tool for finding errors in your configuration files. The start command puts messages in the **.x11startlog** file in your home directory.

If you start X11 and your environment displays as expected, no error messages will be generated and **.x11startlog** will be empty.

However, at some point you may start X11 and your environment does *not* display as expected. For example, maybe one of your terminal windows doesn't display. To view any error messages that occurred, type the following at the command-line prompt in your home directory:

```
more .x11startlog Return
```

Any error messages in the file will be listed on the screen and, although decidedly cryptic in nature, they at least provide a starting place for locating the cause of the error.

Starting a Different Window Manager

The OSF/Motif Window Manager is the default window manager of your X Window System. However, two other window managers are included with the X Window System— **hpwm** (HP Window Manager) and **uwm**. The differences among the three window managers are covered in appendix A.

To use the **hpwm** or **uwm** window manager instead of the OSF/Motif Window Manager, follow these steps:

1. Start your text editor and open the **.x11start** file.
2. Scroll down or search for the line that reads as follows:

   ```
   mwm  $@ & # Start the OSF/Motif Window Manager
   ```

3. Comment out this line by typing a **#** and a space in the left margin.
4. On a new line at the same location, type one of the following commands:

   ```
   hpwm  $@  & # Start hpwm window manager
   ```

   ```
   uwm  $@  & # Start uwm window manager
   ```

5. Save the file and exit the editor.

To put the hpwm or uwm window manager into effect, exit the X Window System by pressing (CTRL) (Left Shift) (Reset). Then restart the window system again.

Starting Programs Automatically

If you'd like to start more than mwm and a single hpterm window when you start X11, you need to add a few more lines to your .x11start file—one line for each client or non-client you want to start.

Syntax and Examples

The syntax for starting a program automatically matches the syntax for running the program from the command-line prompt:

client [*-options*] [&]

Starting Clients

Follow these steps to add other clients to your X11 environment:

1. Start your text editor and open .x11start.
2. Scroll down or search for the line that reads as follows:

```
hpterm -C -geometry 80x24+1+1 $@ &
```

3. On the lines below this, insert command lines for each client you want to start, one client per line.
4. When you're finished, check your syntax and spelling. If all is correct, save the file and exit the editor.

For example, the following two lines start a clock and an hpterm window as part of the initial X environment:

```
xclock -digital -update 10 -geometry 160x30-1+1 &
hpterm -geometry 80x24-1-1 &
```

The first line adds a 160×30 pixel digital clock to the upper right corner of the screen. The clock is updated every 10 seconds. The second line starts an 80 column by 24 line hpterm emulation window in the lower right corner of the

screen. Both clock window and `hpterm` window are the default colors specified in `.Xdefaults` or `/usr/lib/X11/sys.Xdefaults`.

Note that both lines end with an ampersand (`&`), telling the system to start these clients as background processes. Note also that the geometry dimensions of clients like the clock are in pixels; however, the dimensions of terminal windows are in columns (characters across) and lines (characters down).

Starting Non-Clients

Starting non-clients (commands or programs) automatically is similar to starting clients. Follow these steps to add non-clients to your X11 environment:

1. Start your text editor and open `.x11start`.
2. Scroll down or search for the line that reads as follows:

```
hpterm -C -geometry 80x24+1+1 $@ &
```

3. On the lines below this, insert command lines for each non-client you want to start, one non-client per line. Remember that a non-client, because it does not create its own window, is started by the `-e` option (`e` for "execute") from an `hpterm` or `xterm` window.
4. When you're finished, check your syntax and spelling. If all is correct, save the file and exit the editor.

For example, the following two lines start `mailx`, an electronic mail program, and login to a remote host, `hpcvfaa`:

```
hpterm  -e  mailx  &
hpterm  -e  rlogin  hpcvfaa  &
```

Both windows that contain the two non-clients are the default size and colors. Notice also that, in this example, they are *both* at the default location, so first one appears and then the other appears right over it—usually not the best practice. A better way is to include a geometry option for one or both windows. Another alternative is to use a `-iconic` option for one window:

```
hpterm  -iconic  -e  mailx  &
hpterm  -e  rlogin  hpcvfaa  &
```

This modified example starts the `mailx` window as an icon. Only when you want to read mail do you need to change the icon into a window.

Discovering Your Options

The following tables repeat the client option information from chapter 4 so you can avoid excessive page churning caused by flipping back and forth.

Table 5-6. Color Options for Viewable X11 Clients

Option Descriptions		X11 Clients			
To change this ...	**Use this option ...**	**hpterm**	**xterm**	**xclock**	**xload**
Foreground color.	`-fg` *color*	√	√	√	√
Background color.	`-bg` *color*	√	√	√	√
Cursor color.	`-cr` *color*	√	√		
Pointer color.	`-ms` *color*	√	√		
Clock hands color.	`-hd` *color*			√	
Hand edge color.	`-hl` *color*			√	√

Table 5-7. Other Options for Viewable X11 Clients

Option Descriptions		X11 Clients			
To change this ...	**Use this option ...**	**hpterm**	**xterm**	**xclock**	**xload**
Client location.	`-geometry` *w*×*h*±*col*±*row*	√	√	√	√
Font Displayed.	`-fn` *font*	√	√	√	√
Update interval.	`-update` *number*			√	√
Clock chime.	`-chime`			√	
Analog clock.	`-analog`			√	
Start a program.	`-e` *command*	√	√		
Name of icon.	`-n` *name*	√	√		
Title of window.	`-title` *title*	√	√		
Window name.	`-name` *name*	√	√	√	√
Start client as icon.	`-iconic`	√	√	√	√
Where client displays.	`-display` *host:display.screen*	√	√	√	√

You can control the size and location of each viewable client you add to your `.x11start` file using the `-geometry` option. If you don't specify a `-geometry` option, the client appears in the default size and at the default location.

The syntax of the `-geometry` option is as follows:

`-geometry` *width*×*height*[±*x*±*y*]

The size (*width*×*height*) is in characters by lines for terminal windows, and in pixels for clocks and load histograms. The location (±*x*±*y*) is in pixels and depends on the resolution of your screen. Plus values (+) start at the upper left corner of the screen and proceed down and to the right. Minus values (−) start at the lower right corner of the screen and proceed up and to the left.

The following table lists some typical locations for a 1280×1024 high-resolution display.

Table 5-8. Sample Locations for an 80×24 X11 Terminal Window

To position a window here ...	Use this location ...
The upper left corner of the root window.	+1+1
The lower left corner of the root window.	+1−1
The upper right corner of the root window.	−1+1
The lower right corner of the root window.	−1−1
The left side at mid-window.	+1+512
The right side at mid-window.	−1+512
The top of the root window and right of center.	+635+1
Centered at left.	+1+330
Centered at right.	−1+330
Centered in the root window.	+320+330

The options listed here are some of the more commonly used ones. For a complete list of options for each client, see that client's pages in the reference section.

Starting X11 at Login

You can configure your system to start the X Window System at login in two ways:

- Use the HP-UX `SAM` program to add a new account to the system, specifying "X11 Windows" at login.
- Edit your existing `.login` or `.profile` login file to include the `x11start` command.

If your login isn't already configured to start X11 automatically, you can edit your login file to do so. If you are adding a new login to your system, you can use the `SAM` program to tell the system you want X11 at login.

The rest of this section explains how, if you currently have a login on the system, you can edit your `.login` or `.profile` file so that when you log in, your X environment starts automatically.

Modifying Login Files

Which login file you edit depends on which shell command interpreter you use. If you use the C shell, edit `.login`. If you use the Bourne or Korn shell, edit `.profile`.

Finding Out Which Shell You Use

If you are not familiar with which shell you use, type:

```
env Return
```

This command lists your environment variables. Look for the one named SHELL.

Table 5-9. The Environment File for your Shell

If you see ...	Edit ...
`SHELL=/bin/csh`	.login
`SHELL=/bin/sh`	.profile
`SHELL=/bin/ksh`	.profile

Editing the File

Once you have determined the proper login file to edit, use `vi` or some other editor that produces text files to make the following modification to the bottom of the file.

If you use the C shell, follow these steps to modify your `.login` file:

1. Copy your original `.login` to `.login.old` (just in case).
2. Start your text editor and open `.login`.
3. Page or scroll down to the bottom of the file.
4. Insert the following lines at the bottom of the file:

```
if ("`who am i | grep console`" != "" ) then
    exec /usr/bin/x11start
endif
```

5. Save your edited file and exit the text editor.

If you use either the Bourne or the Korn shell, follow these steps to modify your `.profile` file:

1. Copy your original `.profile` to `.profile.old` (just in case).
2. Start your text editor and open `.profile`.
3. Page or scroll down to the bottom of the file.
4. Insert the following lines at the bottom of the file:

```
if [ "`who am i | grep console`" != "" ]
then
    exec /usr/bin/x11start
fi
```

5. Save your edited file and exit the text editor.

These lines verify that you are logging in from the console, and not from a remote location, before starting X11 on your system. This avoids the possibility of undesirable effects caused by inadvertently starting X on your system from a remote login.

Viewing the Result of Your Edit

To view the result of your edit, exit the X Window System by pressing CTRL Left Shift Reset simultaneously. Remember to use the left Shift key.

When the command-line prompt returns to the screen, you can either log out and then log back in, or type `source .login` (if you use the C shell), or type `. .profile` (if you use the Korn shell or Bourne shell) to restart X11.

After a few seconds, your system should start the X Window System. From then on, whenever you login, X11 will start automatically.

Using the 'SAM' Program

You can also start the X Window System automatically for a new user by adding the user with the `SAM` program.

Before you run `SAM`, you must be superuser. As one of the options of "Add a new user", select the X Window System to start at login.

Creating Custom Bitmaps with 'bitmap'

Using the `bitmap` client, you can create your own custom bitmaps and use them to tile your root window, to customize root-window cursor shapes, or to customize menu selections.

Syntax and Options

The syntax for `bitmap` is as follows:

`bitmap` [`-help` | `-display` *host:display.screen* | `-geometry` *w*×*h*±*col*±*row* | `-nodashed` | `-fn` *font* | `-fg` *color* | `-bg` *color* | `-hl` *color* | `-ms` *color* | `-name` *name*] *filename* *Width*×*Height*

-help — Prints a summary of the command usage.

-display — Specifies the screen where `bitmap` is to appear.

-geometry — Sets the size of the `bitmap` window to the specified width×height and locates the window at the specified column and row.

-nodashed	Specifies the `bitmap` grid should use solid lines.
-fn	Specifies the font to use in the `bitmap` command panel labels.
-fg	Specifies the foreground color.
-bg	Specifies the background color.
-hl	Specifies the color of the highlight used to mark the center of a circle (the hot spot) and move areas.
-ms	Specifies the color of the pointer.
-name	Specifies a variable name to use when writing to a bitmap file.
filename	Specifies the name of the bitmap file to open or create.
Width×Height	The size of the bitmap itself. Width and height are measured in pixels with one pixel equal to one cell on the `bitmap` grid.

Using 'bitmap'

The `bitmap` client displays a variable-size grid, a command panel (on the right), and two "preview" bitmaps. You operate `bitmap` by using mouse buttons to "draw" pixels in the grid, one pixel per cell, and by making selections from the command panel. The preview bitmaps enable you to see how your art work looks in regular and reverse video.

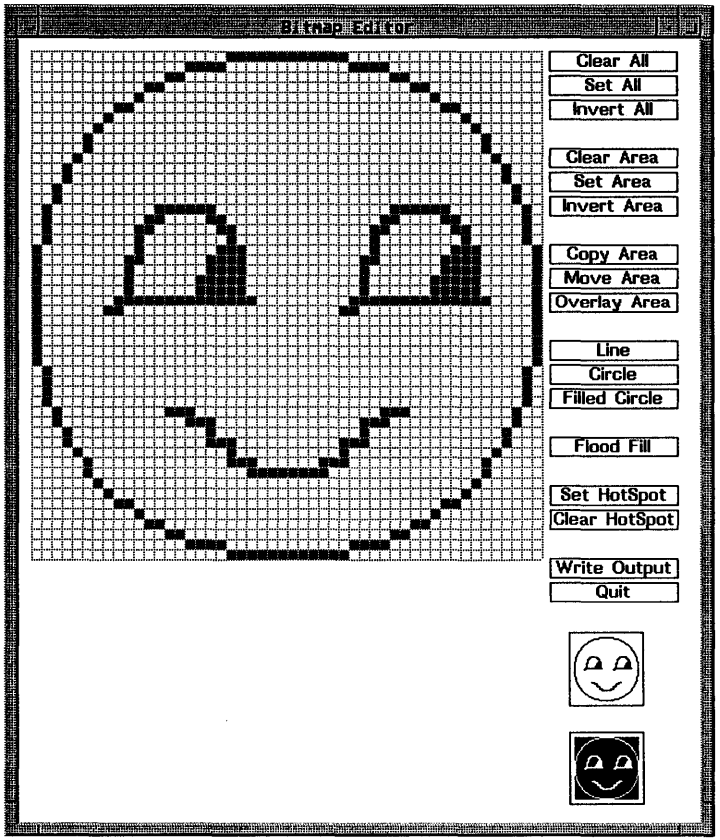


Figure 5-1. The 'bitmap' Client Creates Custom Bitmaps

Currently, `bitmap` uses the button definitions in the following table:

Table 5-10. Mouse Button Definitions for 'bitmap'

If you see ...	On a 2-button mouse press ...	On a 3-button mouse press ...
Button 1	The left button.	The left button.
Button 2	Both buttons simultaneously.	The middle button.
Button 3	The right button.	The right button.

The following table shows how to use the grid portion of the `bitmap` window:

Table 5-11. How to Use the 'bitmap' Grid

If you want to ...	Do this ...
Draw a pixel. Change a cell from background to foreground color.	Click button 1 on the cell.
Invert a pixel color. Change a background colored cell to foreground or a foreground colored cell to background.	Click button 2 on the cell.
Clear a pixel. Change a cell to the background color.	Click button 3 on the cell.

The following table shows how to use the command panel portion of the `bitmap` window:

Table 5-12. How to Use the 'bitmap' Command Panel

If you want to ...	Click button 1 on ...
Set (clear) all cells of the grid to the background color.	Clear All
Set all cells of the grid to the foreground color.	Set All
Set all background colored cells to foreground and all foreground colored cells to background.	Invert All
Set (clear) an area of the grid to the background color.	Clear Area.
Set an area of the grid to the foreground color.	Set Area.
Set any background colored cells in an area of the grid to foreground and any foreground colored cells in that area to background.	Invert Area
Copy one area of the grid to another.	Copy Area
Move an area of the grid to another position.	Move Area
Place one area of the grid over another.	Overlay Area
Draw a line between two points.	Line
Draw a circle with a given center and radius.	Circle
Draw a filled (foreground colored) circle with a given center and radius.	Filled Circle
Fill an enclosed (bounded) area. The area must be completely enclosed.	Flood Fill
Set a "hot spot" to mark the location of the cursor on a cursor bitmap.	Set HotSpot
Erase a "hot spot" from a cursor bitmap.	Clear HotSpot
Save the bitmap to the file specified on the `bitmap` command line.	Write Output
Exit the `bitmap` client.	Quit

Examples

The following examples illustrate some of the possibilities when creating custom bitmaps with `bitmap`.

Creating an Icon Image

You can create a 50 by 50 pixel icon image that you can use for a particular client such as `hpterm` windows. The following bitmap is one example:

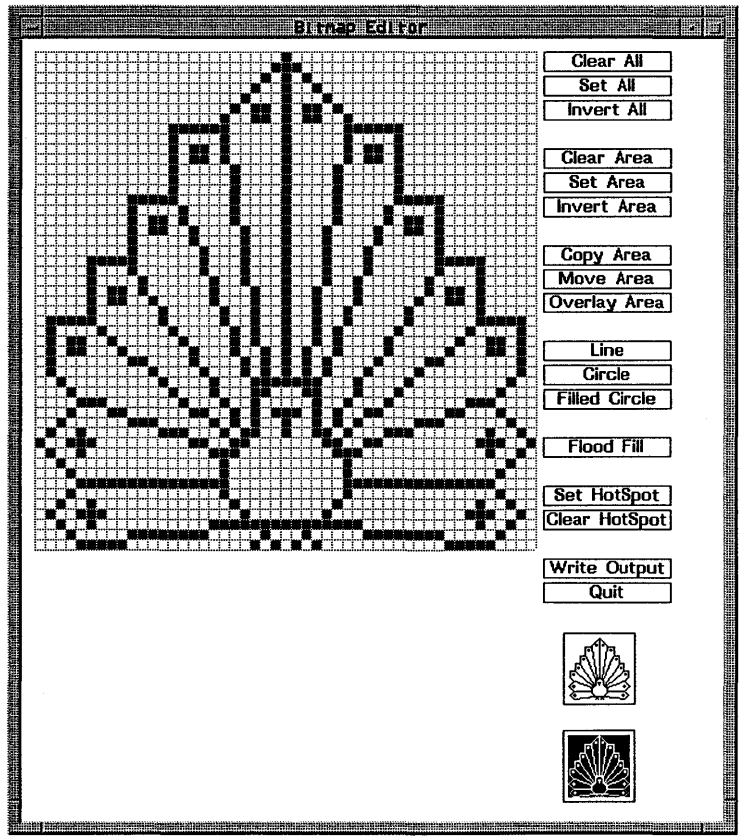


Figure 5-2. A Custom Icon Bitmap

If you name this bitmap "peacock.bits" and keep it in the `~/bits` directory, where ~ stands for the path to your home directory, you can use the bitmap as an image for **hpterm** icons by inserting a line similar to the following in your `.Xdefaults` file:

```
Mwm*HPterm*iconImage:  ~/bits/peacock.bits
```

Whenever you iconify an **hpterm** window, your peacock will appear as the icon.

Creating Root Window Tiles

You can create tiles of any size with which to pattern your root window. One such pattern is the following:

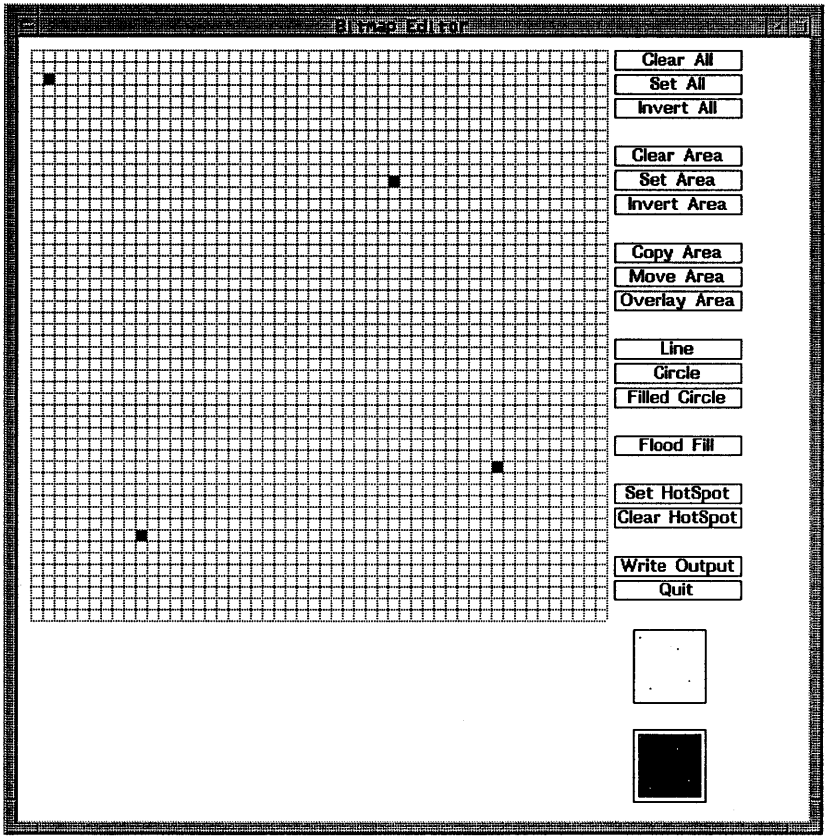


Figure 5-3. A Custom Bitmap for That Spacious Look

This pattern, called for lack of a better name "space.bits," is a random pattern of foreground-colored pixels. Using the `xsetroot` client described shortly, you can use this bitmap for a truly cosmic effect.

Creating Custom Cursors and Masks

Creating a custom cursor (pointer) requires you to make a cursor bitmap and a cursor **mask** bitmap. The mask provides a background for the cursor and prevents the pixels over which the cursor moves from showing through the cursor bitmap.

For example, because the preceding example gave you some space to play with, you might want to create the following cursor, named "shuttle.bits," to help you get from window to window.

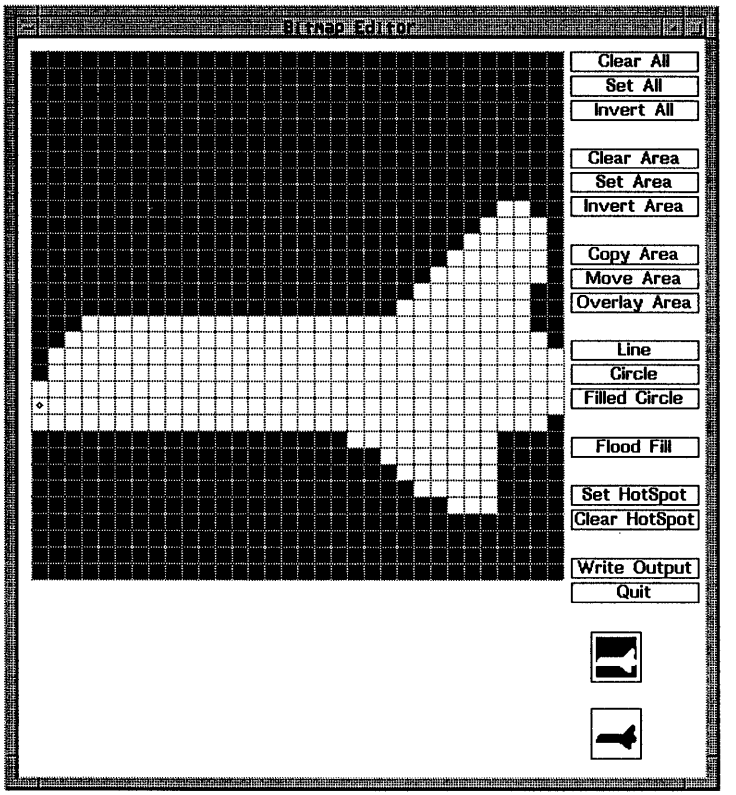


Figure 5-4. A Custom Cursor for Navigating Large Spaces

Note the **hotspot** at the tip of the shuttle's nose. A hotspot is the single pixel that has been designated as the "point" of the pointer.

The following mask, "mask.bits," is made by inverting the original cursor and adding a few extra lines for shading:

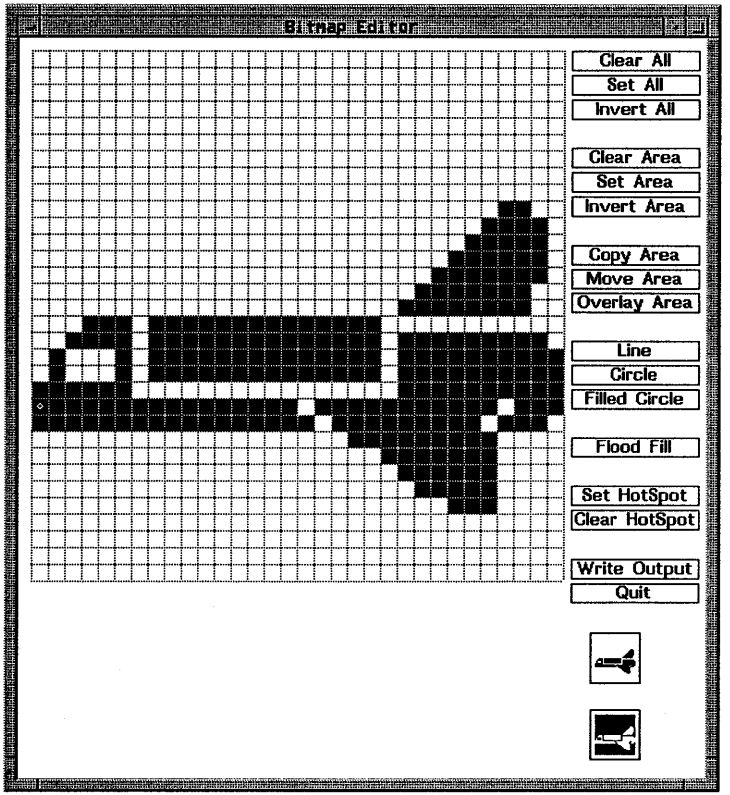


Figure 5-5. A Custom Mask for Navigating Large Spaces

You employ your custom cursor and mask bitmaps using the `xsetroot` client described next.

Customizing the Root Window with 'xsetroot'

The `xsetroot` client enables you to customize the appearance of the root window. You can add color and pattern to the root window, or modify the shape of the cursor when it's in the root window.

Syntax and Options

The `xsetroot` client has the following syntax:

```
xsetroot [ -help
           -def
           -cursor path/cursor path/mask
           -bitmap path/bitmap
           -mod x y
           -gray
           -fg color
           -bg color
           -rv
           -solid color
           -display host:display.screen ]
```

-help	Prints a summary of the command usage.
-def	Resets unspecified root window attributes to their default values.
-cursor	Specifies the cursor bitmap and mask bitmap to use for the root window cursor.
-bitmap	Specifies a bitmap file with which to tile the root window.
-mod	Specifies a modular grid of dimensions *x* by *y* in the foreground color, making a plaid pattern.
-gray	Specifies gray (or grey) for the color of the root window.
-fg	Specifies *color* as the foreground color.
-bg	Specifies *color* as the background color.

-rv	Swaps foreground and background colors.
-solid	Specifies the root window should be colored a solid *color*.
-display	Specifies the host, display number, and screen number of the root window to change.

Examples

The following examples employ the bitmaps created in the last section.

Changing the Root Window Tile Pattern

To change the tile pattern of the root window to a bitmap such as the "space.bits" bitmap, use the following line:

```
xsetroot -bitmap ~/bits/space.bits
```

This line assumes that you keep your bitmaps in a subdirectory of your home directory called `bitmaps`. The actual `xsetroot` command can be issued either from the command line once you've started X or from a line in your `.x11start` file (in which case the changes are made as X11 starts).

Changing the Root Window Cursor

To change the shape of the root window cursor to a bitmap such as the "shuttle.bits" bitmap created above, use the following line:

```
xsetroot -cursor ~/bits/shuttle.bits ~/bits/mask.bits
```

Again, you can issue this line either at the command-line prompt once you've started X or include it as part of your `.x11start` file. Remember, the ~ signifies the path to your home directory.

Working with Fonts

The X Window System includes a variety of **fonts.** A font is a type style, that is, a style in which text characters are printed. For example, the text of most newspapers is printed in the Times Roman font, while the headlines are usually printed in Helvetica.

What Fonts are Available?

Fonts are stored within subdirectories of the `/usr/lib/X11/fonts` directory, as shown in the following diagram.

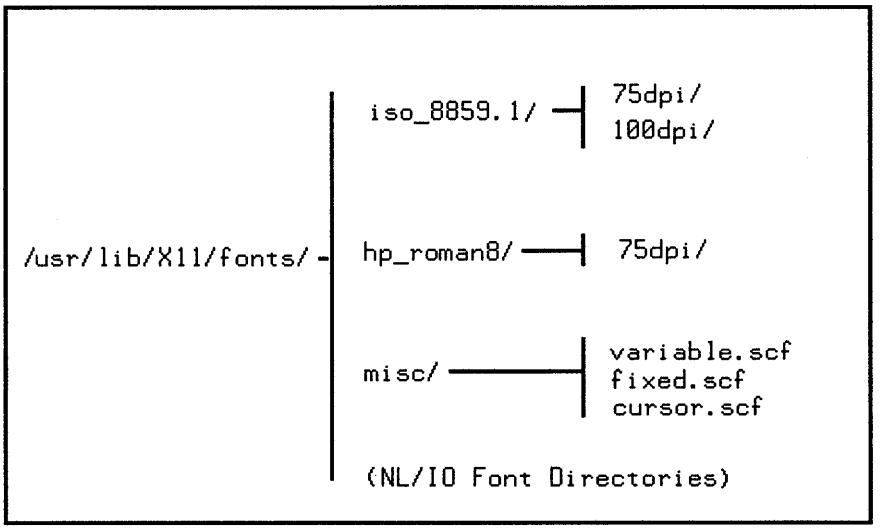


Figure 5-6. The Font Directory Structure

To view what fonts are available in each directory, type:

`ls -p` *directory* Return

Note

The examples in this section use fonts from the `/usr/lib/X11/fonts/hp_roman8/` directory, which was installed with the fileset `X11_FONTA`. If you do not have this directory, these examples will not work for you as written. Find the font directory you *do* have, and use names from that directory instead.

Specifying a Font

Whenever a command or client option calls for *fontname,* you may refer to the font in either of two ways:

- Specify the font's characteristics.
- Provide an "alias" for the font.

Font Characteristics

You may refer to a font by specifying a list of its properties. Any property in the list can be replaced by "*" wild card. Any character in a property can be replaced by a "?" wild card. The server will neither accept nor reject a font based on a particular property if that property is specified by a wild card.

The form of the property string specification is:

"*FontNameRegistry-Foundry-FamilyName-WeightName-Slant*
-SetwidthName-AddStyleName-PixelSize-PointSize-ResolutionX
-ResolutionY-Spacing-AverageWidth-CharSetRegistry -CharSetCoding"

FontNameRegistry	A string that defines the authority that registered the font.
Foundry	A string giving the name of the foundry or font designer.
FamilyName	The trademarked commercial name of the font
WeightName	A string describing the relative weight of the font, such as bold. For human reference only.
Slant	A code indicating whether the font slants to the right, left, or not at all.

	R	Roman
	I	Italic
	O	Oblique
	RI	Reverse italic
	RO	Reverse oblique
SetwidthName	A string describing the width-per-unit of the font, such as compressed or expanded.	
AddStyleName	A string describing anything else needed to uniquely identify the font, such as serif or cursive. Human reference only.	
PixelSize	An integer describing the size of an EM square. An EM square is the size of a box surrounding an M.	
PointSize	An integer giving the EM square size in points (72.27 points = 1 inch)	
ResolutionX ResolutionY	The horizontal (X) and vertical (Y) resolution of the device that the font was designed for, measured in pixels-per-inch.	
Spacing	A code indicating the spacing between units in the font.	
	M	Monospaced (fixed pitch)
	P	Proportional spaced (variable pitch)
	C	Character cell. The glyphs of the font can be thought of as "boxes" of the same width and height that are stacked side by side or top to bottom.
AverageWidth	An integer string giving the average, unweighted width of all the glyphs in the font, measured in 1/10th device-dependent pixels.	
CharSetRegistry	A string identifying the registration authority that registered the specified CharSetEncoding. This is typically the organization and a specific standard number, such as ISO8859 or HP.	

CharSetEncoding — A string identifying the character set fo the specified registry. For example, if CharSetRegistry is ISO8859, then CharSetEncoding "4" identifies the ISO8859.4 character set. If CharSetRegistry is "HP", then CharSetEncoding "roman8" identifies the HPROMAN8 character set.

For example:

```
*-adobe-courier-bold-o-normal-*-10-100-75-75-m-60-hp-roman8
```

specifies a courier, bold, oblique font created by Adobe. The font is 10 pixels tall, 100 tenths of a point tall on a 75dpi×75dpi display. Characters are monospaces, and are an average of 60 tenths of a point wide. Fonts codes are based on the HPROMAN8 encoding.

You may use either upper-case or lower-case letters when you specify a characteristic.

The 'fonts.dir' File

The server associates the font file name and the font property string by means of the `fonts.dir` file. This file is created by the font installation process or by executing the `mkfontdir` utility.

You can view the font characteristics for all the fonts in the directory by typing:

`more` *font directory*`/fonts.dir` Return

If you specify a font using wild cards (* or ?), the server will select the first font in `fonts.dir` that matches the properties that you did specify.

For example, if your `fonts.dir` file looked like this:

```
3
helv008.scf -Adobe-Helvetica-Medium-0-Normal--8-80-75-75-P-47-HP-ROMAN8
helvB008.scf -Adobe-Helvetica-Bold-0-Normal--8-80-75-75-P-48-HP-ROMAN8
helvR08.scf -Adobe-Helvetica-Medium-R-Normal--8-80-75-75-P-46-HP-ROMAN8
```

then if you ask for

```
"*-*-Helvetica-Medium-*-*-*-*-*-*-*-*-*-HP-ROMAN8-*"
```

the first font, `helv008.scf`, will be used.

After the `fonts.dir` file is created or updated, run the following command to inform the server of the change:

```
xset fp rehash [Return]
```

Font Aliases

A font can be referred to by an alias. The alias is shorter and easier to remember (and type) than the complete font description. Aliases are found in the `fonts.alias` file for each directory. A simple `fonts.alias` file is created as part of installing the font.

The `fonts.alias` file provides for two types of aliases:

- File name.

  If the string `"FILE_NAMES_ALIASES"` occurs in the `fonts.alias` file, then the font can be referred to by its file name alone, without the path name or extensions (`.snf` for server natural format or `.scf` for server compressed format).

  Although font names have extensions, usually a `.snf` (server natural format) or `.scf` (server compressed format), you don't have to type the extension when you specify a font.

- A name you select.

  You can specify what alias to use for referring to a font. This helps avoid confusion if you are using fonts with the same name from different directories. You provide the alias name you want to use and the font property string, as shown in the following example.

If this is the `fonts.alias` file in your `/usr/lib/X11/fonts/hp_roman8/75dpi` directory,

```
"FILE_NAMES_ALIASES"
courbold *-adobe-courier-bold-r-normal-*-8-80-75-75-m-50-hp8-roman8
```

then you can refer to the font named `courB08.scf` in any of the following ways:

- `courB08`

The "FILE_NAMES_ALIASES" entry lets you use just the file name.

- `courbold`

  The alias name you specified.

- `*-adobe-courier-bold-r-normal-*-8-80-75-75-m-50-hp-roman8`

  You can always specify the font characteristics, whether or not you have a `fonts.alias` file.

- `*-*-courier-bold-r-normal-*-8-*-*-*-*-*-hp-roman8`

  You can specify enough of the font characteristics to identify the font characteristics you want, and have the rest as wildcards. X11 selects the first font in its search path that matches the specification.

Changing the Alias Search Path

When you specify a font by its alias, by default the server searches the following directories until it finds a match for the alias in one of the directories.

```
/usr/lib/X11/fonts/hp_roman8/75dpi
/usr/lib/X11/fonts/iso_8859.1/75dpi
/usr/lib/X11/fonts/iso_8859.1/100dpi
/usr/lib/X11/fonts/misc
/usr/lib/X11/fonts/hp_kana8
/usr/lib/X11/fonts/hp_japanese/75dpi
/usr/lib/X11/fonts/hp_korean/75dpi
/usr/lib/X11/fonts/hp_chinese_s/75dpi
/usr/lib/X11/fonts/hp_chinese_t/75dpi
```

When the server starts, if any of these directories do not exist, or if a directory does not contain a `fonts.dir` file, that directory is removed from the server's default search path until the server is restarted.

You can check your current font search path by typing:

```
xset q (Return)
```

You can change the directories to be searched by using the `xset` client. (This section covers only the font functions of `xset`, chapter 7 explains the other functions.)

```
xset [ -fp path[,path...]
       fp- path[,path...]
       +fp path[,path...]
       fp+ path[,path...]
       fp default
       fp rehash
       fp= path [,path...]
       q                  ]
```

-fp/fp-	Removes the specified directories from the head (-fp) or tail (fp-) of the font search path.
+fp/fp+	Prefixes (+fp) or appends (fp+) the specified directories to the font search path.
fp= *path*	Specifies the font search path.
fp default	Restores the default font search path.
fp rehash	Causes the server to reread the fonts databases in the current path—done after new fonts are added or deleted, or after `mkfontdir` is run.
q	Display status information, including the current font search path.

Adding or Deleting Fonts

If you add one or more fonts to a directory or delete them from a directory:

1. Run the `mkfontdir` utility program in the directory to which the fonts were added or deleted to update the `fonts.dir` file.
2. Edit the `fonts.alias` file if you want to refer to the font by an alias.
3. Inform the server of the change by typing:

   `xset fp rehash` Return

Choosing Where to Specify a Font

Usually, you specify fonts in the `.Xdefaults` file in your home directory. However, you can specify the font of an individual client (such as the text of a single window) in the command line that starts the client. If you start the client when you start X11, the command line will be in the `.x11start` file. If you start the client from a menu, the command line will be in the `.mwmrc` file.

Making All Instances of a Client Have the Same Font

By inserting a command line in the `.Xdefaults` file in your home directory, you can make every instance of a particular client have the font that you specify.

The syntax for the line is as follows:

*client*fontresource*: *fontname*

The following line in your `.Xdefaults` file changes the font of *every* `hpterm` window to the monospace font `courB08.scf`.

```
HPterm*font: courB08
```

Of course, you can refer to fonts in any of the ways discussed earlier in this chapter. This example uses the file name as an alias.

If you specify a font for the `mwm` window manager, use the `fontList` resource. For example:

```
Mwm*fontList: courB08
```

Specifying the Font of a Window that Starts Automatically

The following line, which uses the standard command-line syntax, in your `.x11start` file overrides any font specification in the `.Xdefaults` file and creates *this particular* `hpterm` window using a font with the alias courB08:

```
hpterm -fn courB08 &
```

Specifying the Font of a Window that Starts from a Menu

The following line, which uses the standard menu selection syntax, in your `.mwmrc` file overrides any font specified in the `.Xdefaults` file and, when you

choose the **New Window** selection from the menu, creates an **hpterm** window using a font with the alias courB08:

```
"New Window"  f.exec "hpterm -fn courB08 &"
```

Displaying a Font with 'xfd'

You can display the complete character set of any valid X Window System font using the **xfd** client.

Syntax and Options

The syntax for **xfd** is as follows:

```
xfd [ -rv
      -fg color
      -bg color
      -bf font
      -tl title
      -in icon
      -icon path/bitmap
      -verbose
      -gray
      -start charnumber
      -geometry parameters
      -display host:display.screen ] -fn fontname
```

-rv — Switches the foreground and background colors (reverse video).

-fg — Specifies the foreground color for **xfd**.

-bg — Specifies the background color for **xfd**.

-bf — Specifies *font* as the font to use for displaying messages at the bottom of the **xfd** window.

-tl — Specifies *title* as the title that should appear in the title bar of the **xfd** window frame.

-in	Specifies *icon* as the name to use for the icon label when an `xfd` client is iconified.
-icon	Specifies the path and filename of the bitmap to use as the icon for the `xfd` client.
-verbose	Displays additional information about a character including: left bearing, right bearing, ascent, descent, and width.
-gray	Specifies a gray background.
-start	Specifies that character number *charnumber* should be the first character displayed (the character in the upper left corner).
-geometry	Specifies the size (width×height) and location (±column±row) of an `xfd` window.
-display	Specifies the host, display number, and screen number on which to display `xfd`.
-fn fontname	Specifies the font to display. If an invalid name, or no specified, you get a "usage" message.

Using 'xfd'

The `xfd` client creates a 16 by 16 grid by default, but you can change the size using the `-geometry` option. Each cell of the grid, starting at the upper left corner, contains a character of the font named on the command line.

Currently, `xfd` uses the button definitions in the following table:

Table 5-13. Mouse Button Definitions for 'xfd'

If you see ...	On a 2-button mouse press ...	On a 3-button mouse press ...
Button 1	The left button.	The left button.
Button 2	Both buttons simultaneously.	The middle button.
Button 3	The right button.	The right button.

Use the following actions to operate the **xfd** client:

Table 5-14. Using the 'xfd' Client

If you want to ...	Do this ...
Page forward to see characters from the specified font that are not currently displayed.	Position the pointer on the **xfd** window and click button 3.
Page backward to see the previously displayed characters.	Position the pointer in the **xfd** window and click button 1.
Display the character set starting with a particular character.	Use the **-start** *charnumber* option on the command line when you start **xfd**.
Show the decimal and hexadecimal value of a character.	Position the pointer in the grid for that character and click button 2.
Show additional information about a character set including left bearing, right bearing, ascent, descent, and width.	Use the **-verbose** option on the command line when you start **xfd**.

Example

The following command line starts an **xfd** window displaying a font with the alias courBO10 in verbose mode. The name of the font appears as a reminder in the title bar.

```
xfd -verbose -tl courBO14 -geometry 300x300-1-1 -fn courBO14 & Return
```

The window has a 300 by 300 pixel size and appears in the lower right corner of the screen. Remember, you can use the full font specification, rather than the alias as shown in the example.

The result of issuing this command line is as follows:

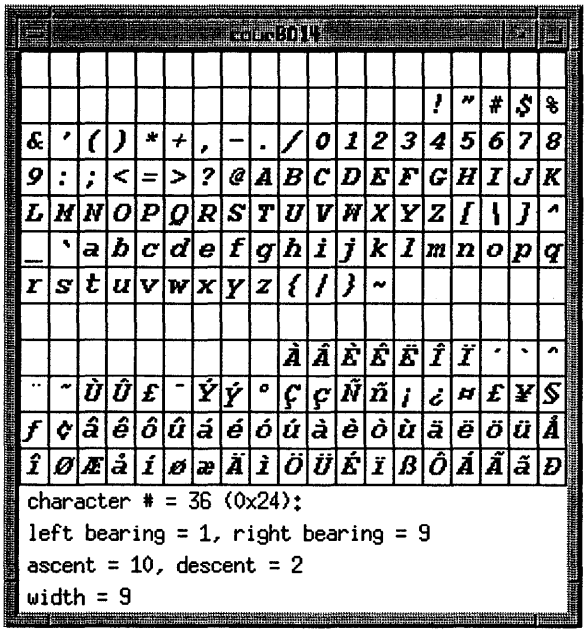


Figure 5-7. CourBO10 Character Set

You can display information about a character by positioning the pointer on that character and clicking button 2. The figure above shows information for the $ character (charnumber = 36) at the bottom of the window.

If the information is too large to be shown in the geometry you specified, you can see other "pages" by positioning the pointer anywhere in the window and clicking button 3.

Using Remote Hosts

Part of the potential of the X Window System is that it enables you to be in two places at once—sort of. You can be logged into your local system working locally and, at the same time be logged into one or more remote hosts.

Gaining Access to Remote Hosts

To gain access to a remote host, you must have the following:

- The address and hostname of the remote host listed in your system's `/etc/hosts` file.
- A valid login (username and password) and home directory on the remote host.

To run clients on the remote system and have them display on your local system:

- The hostname of the remote host listed in a `/etc/X0.hosts` file on your system.

To move files between systems:

- Your system listed in a `.rhosts` file in your home directory *on the remote host.*
- The hostname of the remote host listed in a `.rhosts` file in your home directory *on your local system.*

Setting Up a Login on a Remote Host

To set up a login on a remote host, you need to check that the remote host has a valid internet address and hostname in your system's `/etc/hosts` file, the file that tells the system the address of the other systems on the network.

Also, you need to talk to the system administrator for the remote system. You will need a username, password (if necessary), and a home directory on that system. That way, when you log into the remote host, the remote host will know who you are and where you belong in the directory structure.

Setting Up an 'X0.hosts' File

The remote host must have permission to connect to your display server and display a client program. It gets this permission by being listed in the `/etc/X0.hosts` file on your system. The `X0.hosts` file is an ASCII file that contains the hostnames of all remote hosts that have permission to use your server to display clients on your display screen. Each hostname occupies a separate line as follows:

host1
host2
host3

You can create the file for yourself using any ASCII text editor, or you can use the `xhost` client described below to dynamically add or delete hosts. Changes made with `xhosts` are in effect only for the length of your X session.

Note that the "0" of `X0.hosts` signifies a particular display (combination of screen, keyboard, and mouse) on your system. This is typically the console. If you have another display configuration, you may need another host file. For example, if you are the second display on a system, your host file would probably be `X1.hosts`. A "display" can be either physical (for example, display 0 could correspond to seat 0) or "logical" (for example, if you switch between several configurations, your display could have several logical display numbers, one for each different configuration). For more information, see chapter 7.

Preparing a '.rhosts' File

A `.rhosts` file, placed in your home directory, enables any remote host listed in the file to connect to your system using your login account without having to go through the drudgery of formally logging in and giving a password.

Although this may be convenient to you, it may present an undue opportunity to someone else.

Note

Depending on your situation, a `.rhosts` file could undermine the security of your system and other systems on the network. Check with your system administrator and analyze the security needs of your situation to develop an appropriate plan.

The `.rhosts` file is an ASCII file containing one remote host per line as in the following syntax:

host1 [user]
host2 [user]
host [user]*

To create a `.rhosts` file, you should be in your home directory. Use the following steps:

1. Start your editor and open a file called `.rhosts`.
2. Type the name of the remote host that you want to add.
3. Press Return to move to the next line.
4. Repeat steps 2 and 3 for each remote host you want to add.
5. Check your spelling, save the file, and exit your editor.

If the user name is not included on the same line as the host name, the `.rhosts` file assumes that your remote user name is the same as your local user name.

Adding and Deleting Hosts with 'xhost'

The `xhost` client provides you with a convenient way to dynamically control access to your local system. Using `xhost`, you can add or delete a remote host's permission to access your local X11 display server.

Note that `xhost` only adds or deletes a remote host to or from an internal list created at the start of an X session. It does *not* change the `/etc/X0.hosts` file. To permanently add or subtract access permission you must edit the `/etc/X0.hosts` file using an ASCII text editor such as `vi`.

Syntax and Options

The syntax for `xhost` is as follows:

```
          [ [+]host   ]
          | -host     |
xhost     | +         |
          | -         |
          [ (no option) ]
```

host	Adds *host* to the list of remote hosts with permission to access your local X server.
+host	Adds *host* to the list of remote hosts with permission to access your local X server.
-host	Deletes *host* from the list of remote hosts with permission to access your local X server.
+	Turns off access control, allowing any remote host to access your local X server.
–	Restricts access to your local X server to remote hosts currently listed in your local `/etc/X0.hosts` file.
no option	Prints the list of remote hosts that currently have access to your X server.

You can run `xhost` from the command line at any time you need to change access to your server or to see the current list of remote hosts with access to the server. Remember, changes you make using `xhost` are temporary. They last only as long as your current X session.

Example

The following example allows the remote host `hpcvfgg` to access your local display. As soon as you quit the window system, the access permission is revoked.

```
xhost +hpcvfgg (Return)
```

Starting Programs on a Remote Host

The "Starting Programs" section of chapter 4 covered starting remote clients and non-clients from the command line. You can, however, start remote programs without typing a lengthy command after the command-line prompt.

Starting a Remote Program when you start X11

One way to start a remote program, either a client or non-client, is to start the program when you first start X11. This enables you to have the remote program as a part of your initial environment.

To start a remote client when you start X11, you need to edit the `.x11start` file in your home directory to include one line for each remote client you want to start. The lines are similar to the following:

`remsh` *host* `-n /usr/bin/X11/`*client* `-display` *host:display.screen* [`&`]

Here *host* is the name of the remote host. The *client* can be any X client. And the `-display` option specifies the system, display number, and screen number where the client is to display, typically your local system.

To start a remote non-client when you start X11, edit your `.x11start` file to include one line for each remote non-client. The line begins by starting a remote shell (`remsh`), then a terminal emulation window in which to run the non-client, and finally the non-client:

`remsh` *host* `-n /usr/bin/X111/hpterm -display` *host:display.screen* `-e` *non-client* [`&`]

The `-e` option ("e" for execute), when used with an `hpterm` or `xterm` client, executes a command, in this case the non-client.

Note that an alternate syntax is to start an `hpterm` window and use the `-e` option to execute a remote login (`rlogin`) that makes the window a terminal of the remote host.

For example, the following lines start a remote login (non-client) and a remote load histogram (client) on the host `hpcvfaa` and display the results on the console of the local system, `hpcvfbb`:

```
remsh hpcvfaa -n /usr/bin/X11/xload -display hpcvfbb:0.0 &
hpterm -title "hpcvfaa login" -e rlogin hpcvfaa &
```

Starting a Remote Program from a Menu

Starting a remote program from a menu requires editing the `.mwmrc` to include the proper line to start the program. The process is similar to starting the program from `.x11start`.

Use a line similar to the following to start a remote client:

selection `f.exec "remsh` *host* `-n /usr/bin/X11/`*client* `-display` *h:d.s* `&"`

To start a remote non-client, use the above syntax, adding a `-e` option as the last option before the `&`. Alternately, create an `hpterm` window and use `-e rlogin` *host* to start a remote login.

The explanation of this syntax is the same as the syntax used in `.x11start` with the exception of *selection,* the selection that appears on the menu, and `f.exec`, the OSF/Motif Window Manager function that starts a process, in this case an `hpterm` window.

Example

The following example starts a login on remote host `hpcvfaa`. The login process is initiated by choosing the "hpcvfaa Login" selection from the root menu.

```
# Root Menu Description
Menu  DefaultRootMenu
{
"Root Menu"      f.title
"New Window"     f.exec "hpterm &"
"hpcvfaa Login" f.exec "hpterm -e rlogin hpcvfaa &"
"Shuffle Up"     f.circle_up
"Shuffle Down"   f.circle_down
"Refresh"        f.refresh
no-label         f.separator
"Restart"        f.restart
}
```

Where To Go Next

This chapter has discussed customizing the operation of your window system environment to suit your personal needs. There is additional customization that you can perform beyond what was presented here. Some of it is a little more difficult to comprehend and it would be a good idea to consult with your system administrator before attempting to implement some of the changes.

If you are satisfied with the current look and performance of your window system environment, you may want to stop here, use the system for a few days or weeks, and then perhaps "fine tune" it based on your experience.

On the other hand, if you are interested in more extensive customizations to the OSF/Motif Window Manager, in special environment configurations, in printing, or in graphics, you should read chapters 6, 7, 8, and 9 respectively.

If your interest is in programming, turn to one of the programming manuals.

6

Managing Windows

Managing windows is the job of the window manager. This chapter begins by briefly mentioning the clients related to window management. But most of the chapter discusses the nitty-gritty details of how to use the OSF/Motif Window Manager (`mwm`), its resources, and functions to manage your window environment.

It is *not* necessary to read this chapter to use the window manager or X, but if your management needs go beyond adding and deleting menu selections, browsing this chapter should prove interesting. After discussing the clients, the chapter reviews some familiar aspects of window control, but becomes more technical once these basics have been covered.

The chapter organizes window manager resources and functions into the following task-oriented topics:

- Managing the general appearance of window frames.
- Working with icons.
- Managing window manager menus.
- Using the mouse.
- Using the keyboard.
- Controlling window size and placement.
- Controlling resources with focus policies.
- Adding mattes to client windows.

Clients That Help You Manage Windows

Of the clients listed in the reference section of this manual, six are directly related to window management:

- `resize`
- `xrefresh`
- `xwininfo`
- `mwm`
- `uwm`
- `hpwm`

Resetting Environment Variables with 'resize'

The `resize` client resets three environment variables: TERM, LINES, and COLUMNS. This enables a shell to reflect the current size of its window.

Don't confuse `resize`, the client, with `f.resize` the window manager function. The `f.resize` function changes the size of a window, but does not reset any environment variables. The `resize` client, on the other hand, does not change the size of a window, but it does reset the environment variables. Resetting the environment variables enables non-client programs to adjust their output to the window's new size.

When to Use 'resize'

Use `resize` whenever you resize a terminal emulator window and want a non-client program running in that window to reflect the window's new size. The `resize` client is typically used as an argument to the HP-UX `eval` command.

Syntax and Options

The syntax for `resize` is as follows:

```
resize { -c
       { -h
       { -s [row col]
       { -u
       { -x
```

-c	Resets the environment variables for **csh** shells.
-h	Uses Hewlett-Packard terminal escape sequences to determine new window size.
-s	Uses Sun escape sequences to determine new window size. New row and column sizes are specified with *row* and *col. col*
-u	Resets the environment variables for **sh** and **ksh** shells.
-x	Uses VT102 escape sequences to dermine new window size.

Example

To see what the current COLUMN and LINES settings are, type the following command:

```
resize (Return)
```

After you have resized a window either by dragging the window frame or by choosing the "Size" selection from the window menu, you can reset the LINES, and COLUMN environment variables to reflect the new window size by issuing the following command:

```
eval 'resize' (Return)
```

If you find yourself typing the above command too often, you can make things a little easier on yourself. If you use **csh**, try using an alias. The following line in your **.cshrc** file enables you to run **resize** by typing **xr**.

```
alias xr 'set noglob; eval 'resize''
```

If you use **sh** or **ksh** create an **xr** function like the following:

```
xr()  {eval 'resize';}
```

Repainting the Screen with 'refresh'

The **xrefresh** client "repaints" the screen or a specified portion of the screen. It does this by mapping, then immediately unmapping, a window over the area to be repainted. This obscuring-unobscuring causes the area to be redrawn. Repainting a screen removes the "graphics litter" that occasionally disfigures a screen.

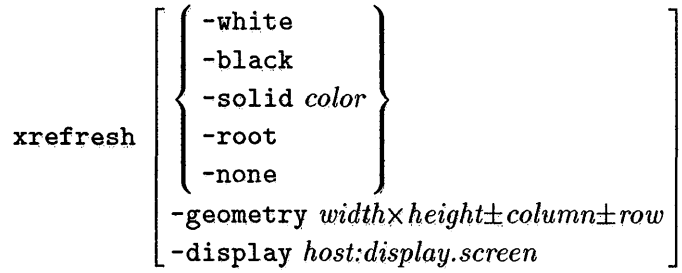
The **xrefresh** client performs a similar task to the **f.refresh** window manager function. However, the **xrefresh** client, because of its options, is more versatile.

When to Use 'xrefresh'

You can use **xrefresh** from the command line of any terminal window and, using the **-display** option, you can repaint any display.

Syntax and Options

The syntax for **xrefresh** is as follows:



-white — Uses a white window to map the screen.

-black — Uses a black window to map the screen.

-solid — Uses a *color* colored window to map the screen.

-root — Uses the root window to map the screen.

-none — Uses a transparent window to map the screen (default).

-geometry — Repaints a *width*×*height* region located at ±*column*±*row* on the screen (dimensions are in pixels).

-display Specifies the screen to refresh.

Example

The following example illustrates using **xrefresh** from the command line to repaint the upper left quarter of the screen.

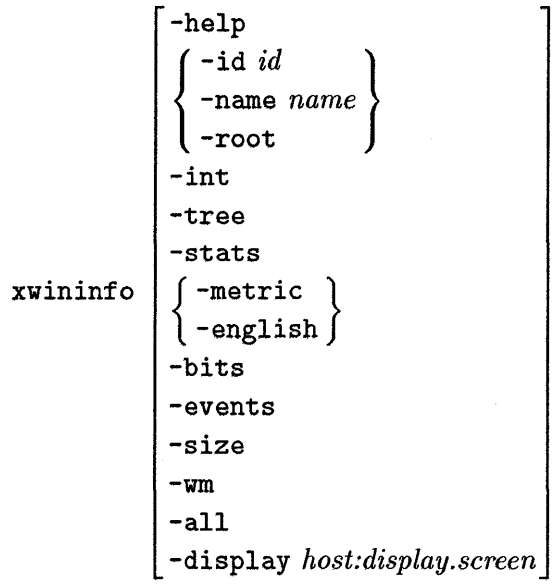
```
xrefresh -white -geometry 800x400+1+1
```

Getting Window Information with 'xwininfo'

The **xwininfo** client is a utility program that displays useful information about windows.

Syntax and Options

The syntax for **xwininfo** is as follows:



-help Prints a summary of the command usage.

-id Specifies the target window by window id.

-name Specifies the target window by name.

-root	Specifies the root window as the target.
-int	Displays window information, normally shown as hexadecimal, as decimal.
-tree	Displays ids and names of the root, parent, and child windows.
-stats	Displays window id, location, size, depth, and other information as hexadecimal.
-metric	Displays height, width, x and y information in millimeters.
-english	Displays height, width, x and y information in inches, feet, yards.
-bits	Displays information about bit and storage attributes.
-events	Displays event masks of the target window.
-size	Displays sizing information about the target window.
-wm	Displays the window manager hints for the target window.
-all	Displays all available information about a window.
-display	Specifies the host, display, and screen to target.

Example

This example illustrates the result of issuing the following command:

```
xwininfo -stats Return
```

Once you issue the command, select a window as the target of your inquiry by moving the pointer into that window and clicking button 1.

```
xwininfo ==> Window id: 0x200013 (hpcvxRW)
  ==> Upper left X: 6
  ==> Upper left Y: 6
  ==> Width: 484
  ==> Height: 316
  ==> Depth: 8
  ==> Border width: 4
  ==> Window class: InputOutput
  ==> Colormap: 0x80065
  ==> Window Bit Gravity State: NorthWestGravity
```

```
==> Window Window Gravity State: NorthWestGravity
==> Window Backing Store State: NotUseful
==> Window Save Under State: no
==> Window Map State: IsViewable
==> Window Override Redirect State: no
==> Corners:  +6+6  -782+6  -782-694  +6-694
-geometry =80x24+6+6
```

Managing Windows with the OSF/Motif Window Manager

The OSF/Motif Window Manager (mwm) is an X11 client that manages the appearance and behavior of objects on the root window. You control **mwm** and its management operations using a mouse, a keyboard, and a functional window frame similar to Microsoft's Presentation Manager. Additionally, **mwm** has a root menu to assist you in the general control of the root window.

The mwm client receives configuration information from three files: `/usr/lib/X11/sys.Xdefaults`, `/usr/lib/X11/system.mwmrc`, and `/usr/lib/X11/app-defaults/Mwm`. You can copy the first two of these files to your home directory, as `.Xdefaults` and `.mwmrc` respectively, and edit them to create a window manager that exactly fits your needs.

How to create your own personal window manager is the subject of the rest of this chapter.

When to Use 'mwm'

The OSF/Motif Window Manager is the default window manager for your X Window System. It is started from `$HOME/.x11start` when you start X11. If that file doesn't exist, mwm is started from `/usr/lib/X11/sys.x11start`.

Syntax and Options

The syntax for mwm is as follows:

```
mwm [-display host:display.screen]
    [-xrm resourcestring          ]
```

-display Specifies the screen to use.

-xrm Specifies using the named resource on starting.

Example

The following line in `.x11start` in your home directory starts `mwm`.

```
mwm $@ &
```

The `$@` passes the window manager options specified on the `x11start` command line.

Managing Windows with Other Window Managers

The `hpwm` (HP Window Manager) and `uwm` clients provide an alternative to managing windows with the OSF/Motif Window Manager.

Appendix A summarizes the differences between `mwm`, `hpwm`, and `uwm`.

Managing the General Appearance of Window Frames

In chapter 5, you read about `/usr/lib/X11/sys.Xdefaults` and `.Xdefaults`.

The `sys.Xdefaults` file is the system file that controls the X environment of users who don't have a `.Xdefaults` file in their home directory. `.Xdefaults` overrides the system-wide effects of `sys.Xdefaults`, enabling you to customize your own environment while not interfering with the environments of others.

By editing `.Xdefaults`, you can control the general appearance of the window frames in your environment. If you are a system administrator, you can control the system-wide general appearance of window frames by editing `/usr/lib/X11/sys.Xdefaults`.

Three aspects of the general appearance of window frames are under your control.

Color	The color of foreground, background; and top, bottom, and side shadows.
Tile	The mixture of foreground and background color that composes the pattern of the frame surface.

Font — The style (including size) of the text characters in the title bar, menus, and icon labels.

To control color, tile (pixmap) pattern, or fonts, you specify a value for the appropriate window manager **resource.** A resource controls an element of appearance or behavior. Resources are always named for the elements they affect.

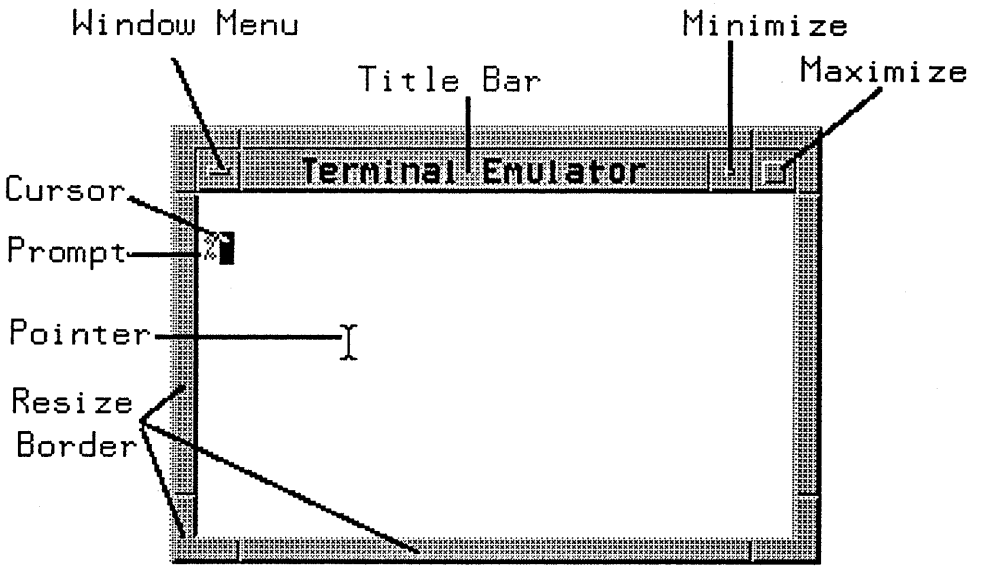


Figure 6-1. A OSF/Motif Window Manager Frame Showing Frame Elements

For example, suppose you want to color the background of your window frame (an element of appearance) Firebrick red. Edit `.Xdefaults`, making `Mwm*background:` (the resource controlling the background color of the frame) the color Firebrick (a color value). The line in `.Xdefaults` would read as follows:

```
Mwm*background: Firebrick
```

Coloring Window Frames

You can use any of the standard X11 colors listed in `/usr/lib/X11/rgb.txt` to color frame elements. In addition, you can create your own colors using hexadecimal values (see "Customizing the Color of Clients" in chapter 5). Frame elements and resources exist for inactive windows (any window not having the current keyboard focus) and for the active window (the window having the current keyboard focus). This enables you to distinguish the active window by giving it special "active window" colors.

Coloring Individual Frame Elements

The following table lists the individual elements of inactive and active window frames, and the resources that control their color, for the OSF/Motif Window Manager.

The default settings provide a 3-D visual effect without you having to specify the exact colors for every frame element.

Table 6-1. Window Frames Resources for a Color Display

To color this ...	Use this resource ...	The default value is ...
Background of inactive frames.	`background`	LightGrey
Left and upper bevel of inactive frames.	`topShadowColor`	Lightened `background` color
Right and lower bevel of inactive frames.	`bottomShadowColor`	Darkened `background` color
Foreground (title bar text) of inactive frames.	`foreground`	Darkened `bottomShadowColor`
Background of the active frame.	`activeBackground`	CadetBlue
Left and upper bevel of the active frame.	`activeTopShadowColor`	Lightened `activeBackground` color
Right and lower bevel of the active frame.	`activeBottomShadowColor`	Darkened `activeBackground` color
Foreground (title bar text) of the active frame.	`activeForeground`	Darkened `activeBottomShadowColor`

Example

The following lines in the `.Xdefaults` file in your home directory give the window manager frame a maroon foreground and a gray background. The background color is used to generate colors for the top and bottom shadow elements so that a 3-D effect is achieved.

The 3-D effect is useful in providing a quick visual indication of selected items, the active window, and so on.

```
Mwm*foreground: Maroon
Mwm*background: Gray
```

Changing the Tiling of Window Frames With Pixmaps

A **pixmap** is a way of creating shades of colors. Each pixmap is composed of **tiles.** A **tile** is a rectangle that provides a surface pattern or a visual texture by "mixing" the foreground and background colors into a color pattern. The concept is analogous to using ceramic tiles to provide a floor or countertop with a pattern or texture.

Generally, the fewer the number of colors your display can produce, the more important tiling will be to you. For example, if you had a monochrome display (two colors—black and white), you could tile the window frames of your X environment in shades of gray to achieve a 3-D look.

The OSF/Motif Window Manager has resources that enable you to tile the frame background and bevels for both inactive and active windows.

Table 6-2.
Tiling Window Frames with Window Manager Resources

To tile this ...	Use this resource ...	The default for color displays is ...
Background of inactive frames.	`backgroundPixmap`	NULL
Right and lower bevels of inactive frames.	`bottomShadowPixmap`	NULL
Left and upper bevels of inactive frames.	`topShadowPixmap`	NULL
Background of the active frame.	`activeBackgroundPixmap`	NULL
Right and lower bevels of the active frame.	`activeBottomShadowPixmap`	NULL
Left and upper bevels of the active frame.	`activeTopShadowPixmap`	NULL

The following table lists the acceptable values for pixmap resources:

Table 6-3. The Values to Use for Tiling Window Frames

To tile an element this color ...	Use this value ...
The foreground color.	foreground
The background color.	background
A mix of 25% foreground to 75% background.	25_foreground
A mix of 50% foreground to 50% background.	50_foreground
A mix of 75% foreground to 25% background.	75_foreground
In horizontal lines alternating between the foreground and background color.	horizontal_tile
In vertical lines alternating between the foreground and background color.	vertical_tile
In diagonal lines slanting to the right alternating between the foreground and background color.	slant_right
In diagonal lines slanting to the left alternating between the foreground and background color.	slant_left

The following figure illustrates the valid tile values:

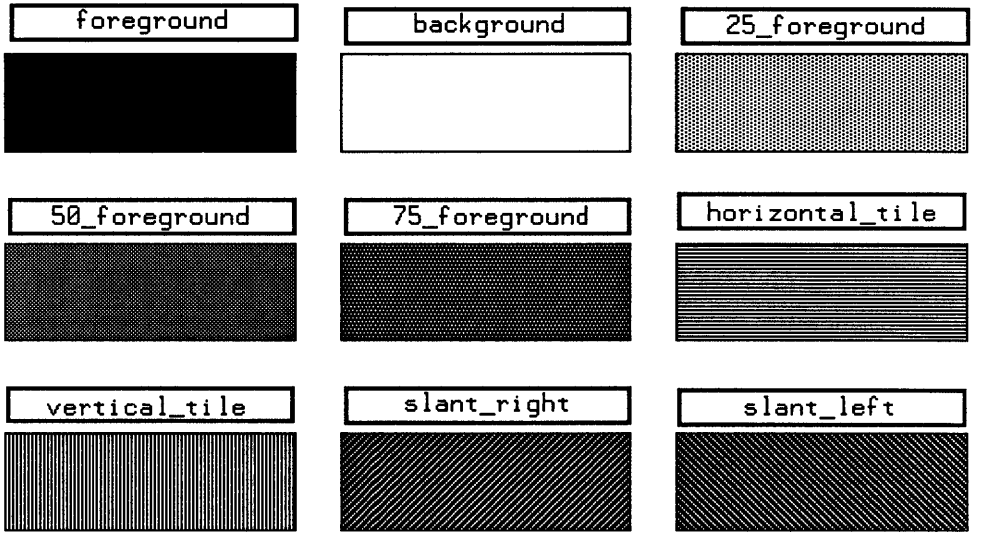


Figure 6-2. Valid Tile Values

Frame Resources For Monochrome Displays

If **mwm** determinges that the monitor is monochrome, and no color resources are specified for frame elements, **mwm** uses defaults appropriate for monochrome displays. **Mwm*background** and **Mwm*activeBackground** are set to White. The following table lists the frame elements, resources, and defaults for monochrome monitors.

Table 6-4.
Window Frame Resource Values for Monochrome Monitors

The background is ...	For this resource ...	The default value is ...
White	`topShadowColor`	White
White	`bottomShadowColor`	Black
White	`foreground`	Black
White	`topShadowPixmap`	foreground
White	`activeBackgroundPixmap`	foreground
White	`activeTopShowdowPixmap`	50_foreground

The `sys.Xdefaults` file contains a set of entries that provides a more attractive window shading for monochrome displays. These entries start with `mwm_bw`, and require that you start `mwm` with the name `mwm_bw`. To do this, edit the following line in `.x11start`:

```
mwm & #Starts the mwm window manager
```

to read:

```
mwm -name mwm_bw & #Starts the mwm window manager
```

You must restart X11 in order for this change to take effect.

Specifying a Different Font for the Window Manager

The default font for the text of the OSF/Motif Window Manager is the `fixed` font. However, you can use the `fontList` resource to specify a different font if you desire. The `fontList` resource can use any valid X11 font name as its value. For more information about fonts, see "Working With Fonts" in chapter 5.

The Syntax for Declaring Resources

The above general appearance resources for the OSF/Motif Window Manager and their values are specified in `sys.Xdefaults` (system-wide) or `.Xdefaults` (your personal environment). The syntax you use differs depending on whether you want the resource to control the general appearance of an element or the general appearance of that element *for a particular object.*

For example, the syntax you use to specify a frame background of Wheat is different from the syntax you use to specify that only menus have a background of Wheat.

The Syntax for the General Appearance of Elements

Use the following syntax in `sys.Xdefaults` or `.Xdefaults` to specify the general appearance of frame elements:

`Mwm*`*resource*`:` *value*

For example, if you want the foreground and background of inactive window frames to be the opposite of the foreground and background of the active window frame, and you choose the colors SteelBlue for background and VioletRed for foreground, you would have the following lines in your `.Xdefaults` file.

```
Mwm*background:          SteelBlue
Mwm*foreground:          VioletRed
Mwm*activeBackground:    VioletRed
Mwm*activeForeground:    SteelBlue
```

The Syntax for Window Frame Elements of Particular Objects

You can specify the general appearance of window frame elements for three particular objects.

- Menus (includes *both* system and root menus).
- Icons (includes the frame elements of *all* icons).
- Clients (includes the frame elements of *all* clients).
- Feedback (window manager feedback windows).

This gives you the ability to select a different color or font for a particular object, perhaps menus, while the other objects (icons and fonts) remain the same. To do this, use the following syntax:

```
Mwm*{ menu     }*resource: value
    { icon     }
    { client   }
    { feedback }
```

For example, if you want the general appearance of the clients in your environment to be SteelBlue and VioletRed, but want your menus to be different, you could add the following lines to `.Xdefaults`.

```
Mwm*background:          SteelBlue
Mwm*foreground:          VioletRed
Mwm*activeBackground:    VioletRed
Mwm*activeForeground:    SteelBlue

Mwm*menu*background:     SkyBlue
Mwm*menu*foreground:     White
```

Working with Icons

Icons provide a handy way to straighten up a cluttered workspace. They also provide you with a great tool for efficient multi-processing. For example, you could open several windows, start processes in each, and then iconify them all—letting the processes run their individual courses while you sit back and read your electronic mail and work in an editing window.

Studying Icon Anatomy

Like the other objects that appear on the root window, you can configure the appearance of all icons in `sys.Xdefaults`, for system-wide icons, or `.Xdefaults`, for your own personal icons. Icons consist of two parts:

- A text label.
- A graphic image.

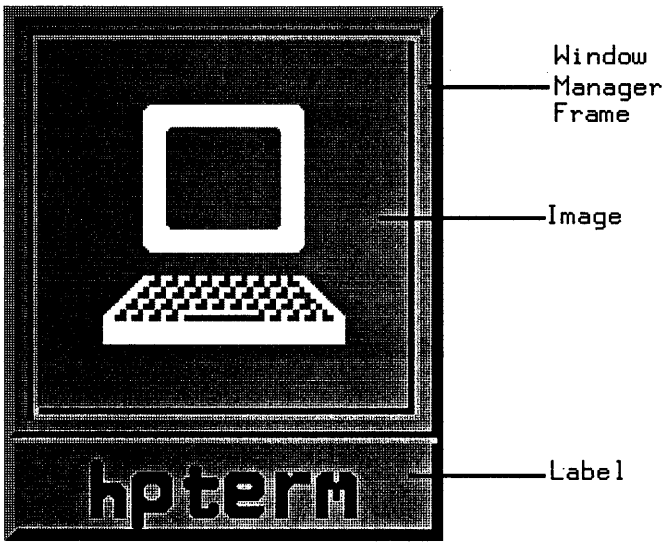


Figure 6-3. An Icon Has Two Parts

The Label

An icon **label** is the text beneath an icon image. A label is usually supplied by the client (via the `WM_ICON_NAME` window property), but some clients, for example `hpterm` and `xfd`, provide a command-line option enabling you to write in your own label.

Icon labels are truncated on the right to the width of the icon image, so if you use small images, don't get too windy with your labels.

The Image

An icon image (a bitmap) is the actual graphic illustration of the icon. An image can come from any one of the following three sources:

client	A client can use the `WM_HINTS` window property to specify either an icon window or a bitmap for the window manager to use as the icon image.
user	You, the user, can specify an icon image using the `iconImage` resource.

default — The window manager will use its own built-in default icon image if an image is not specified elsewhere.

The window manager uses the following default order of precedence in choosing an icon image:

1. A specific user-supplied icon image resource.
2. A client-supplied icon image.
3. A default icon image.

The resource `useClientIcon` lets you interchange the precedence of user-supplied icon images and client-supplied icon images. The default value is "False." When the resource is set to "True," client-specified icon images have precedence over user-supplied icon images.

Manipulating Icons

You manipulate icons similar to the way you manipulate windows, by positioning the pointer on the icon and clicking, double-clicking, or dragging a mouse button (depending on what you want to happen). You can also use icons in situations where you want to start several processes when you start X11, but don't want to clutter your screen with windows you won't immediately use; simply start the processes as icons.

Operating on Icons

The following table lists the operations you can perform on icons:

Table 6-5. You Can Manipulate Icons in These Ways

To do this ...	Position the pointer on the icon and ...	What this does is ...
Turn an icon into a window.	Double-click button 1.	Restores the window to its former size and location.
Move an icon around on the root window.	Drag button 1.	Moves a wire frame with the pointer showing where the icon will be moved.
Give an icon keyboard input focus	Press button 1.	Makes the icon the focus of keyboard input.
Move an icon to the top of the window stack.	Click button 1 on an icon that has keyboard input focus.	Moves a partially concealed icon to the front of the root window.
Select an icon and display its window menu.	Click button 1, or press Shift Esc or Alt space.	Gives an icon keyboard focus and displays the icon's window menu. The window menu for an icon is exactly like the window menu of its associated window. No window is active while the icon has the keyboard focus.

Starting Clients as Icons

You can start clients as icons when you start X11. This gives you the benefit of having the client only a double-click away, while not cluttering your display with windows you're not using.

Some clients have iconify options, like `hpterm`'s `-iconic` option. As you start the client from a command line in your `.x11start` or `.mwmrc` file, adding the iconify option to the line enables you to start the client but to display it

initially as an icon. Later, when you're ready to use the client, you double-click on the icon and you're ready to go.

Controlling Icon Placement

By default, the window manager places icons in the lower left corner of the root window. Successive icons are placed in a row proceeding toward the right. Icons are prevented from overlapping. An icon will be placed in the position it last occupied if no icon is already there. If that place is taken, the icon will be placed at the next free location.

The following three resources enable you to control the placement of icons:

Table 6-6.
Controlling Icon Placement with Window Manager Resources

To specify this ...	Use this resource ...	The default value is ...
A placement scheme for icons.	`iconPlacement`	left bottom
The distance between screen edge and icons.	`iconPlacementMargin`	the default space between icons
Automatic icon placement by the window manager.	`iconAutoPlace`	True

Changing Screen Placement

You can place icons or you can have the window manager do it for you. The window manager will place icons automatically, based on the placement scheme you specify with the `iconPlacement` resource, if you give `iconAutoPlace` a value of "True." If you would rather determine icon placement without help from the window manager, give `iconAutoPlace` a value of "False."

The following table lists the icon placement schemes available to you:

Table 6-7. Schemes for Automatic Placement of Icons

If you want this icon placement ...	Choose this scheme ...
From left to right across the top of the screen.	left top
From right to left across the top of the screen.	right top
From left to right across the bottom of the screen.	left bottom
From right to left across the bottom of the screen.	right bottom
From bottom to top along the left of the screen.	bottom left
From bottom to top along the right of the screen.	bottom right
From top to bottom along the left of the screen.	top left
From top to bottom along the right of the screen.	top right

The Syntax for Icon Placement Resources

The resources that place icons share a common syntax:

`Mwm*`*resource value*

For example, if you want automatic placement of icons starting at the top of the screen and proceeding down the right side, you would have the following lines in your `.Xdefaults` file:

```
Mwm*iconPlacement: top right    Specifies the placement scheme.
Mwm*iconAutoPlace: True         Specifies automatic placement.
```

Controlling Icon Appearance and Behavior

The OSF/Motif Window Manager offers you a number of resources to control the specific appearance and behavior of icons. Among these are resources that enable you to select icon decoration, control icon size, and create new icon pixmaps.

Selecting Icon Decoration

Using the `iconDecoration` resource, you can select exactly what parts of an icon you want to display:

Table 6-8. The Values That Control the Appearance of Icons

If you want an icon that looks like this ...	Use this value ...
Just the label.	label
Just the image.	image
Both label and image.	label image
The label of an active icon isn't truncated.	label activelabel

Sizing Icons

Each icon image has a maximum and minimum size. The OSF/Motif Window Manager has both default sizes as well as maximum and minimum allowable sizes.

Table 6-9. The Maximum and Minimum Sizes for Icon Images

	Maximum Size	Minimum Size
Default	50×50 pixels	32×32 pixels
Allowable	128×128 pixels	16×16 pixels

If you plan to do a lot of work with icons, remember to keep your images within the maximum and minimum limits. How the window manager treats an icon depends on the size of the image in relation to the maximum and minimum sizes.

Table 6-10. Icon Size Affects Icon Treatment

If an icon image is ...	The window manager will ...
Smaller than the minimum size.	Act as if you specified no image.
Within maximum and minimum limits.	Center the image within the maximum area.
Larger than the maximum size.	Clip the right side and bottom of the image to fit the maximum size.

You can use the following two resources to control icon image size:

Table 6-11.
Controlling Icon Image Size with Window Manager Resources

To specify this ...	Use this resource ...
Maximum size of an icon image.	`iconImageMaximum`
Minimum size of an icon image.	`iconImageMinimum`

If you figure icon size based on how much screen "real estate" you can afford to devote to icon space, bear in mind that the overall width of an icon is the image width *plus* border padding and the image height is the icon height *plus* border padding.

Using Custom Pixmaps

When you iconify a client, either the client supplies its own icon image, the window manager supplies a default image, or you supply an image of your own.

You will obtain some icon images as "ready-made" bitmaps. At other times, you may want to use the `bitmap` client (discussed in chapter 5) to create one of your own. In either case, to use your bitmap, you only need to tell the window manager where the bitmap is located.

To tell the window manager to use a particular bitmap for an icon image, use the `iconImage` resource. The value that follows this resource is the path to the bitmap file you want to use. Note that, if specified, this resource overrides any client-specified image.

You also have the ability, using the `bitmapDirectory` resource, to direct the window manager to search a specified directory for bitmaps. The `bitmapDirectory` resource causes the window manager to search the specified directory whenever a bitmap is named with no complete path. The default value for `bitmapDirectory` is `/usr/include/X11/bitmaps`.

The Syntax for Resources that Control Icon Appearance

The resources that control icon appearance have the following syntax:

`Mwm*`*resource*`:` *value*

For example, you could use `bitmapDirectory` to search a `bitmap` subdirectory in your home directory for custom bitmaps by inserting the following line in your `.Xdefaults` file:

`Mwm*bitmapDirectory: /users/`*yourusername*`/bitmap`

The `iconImage` resource has three other syntaxes. The syntax you should use depends on which of the following statements is true:

Table 6-12. The 'iconImage' Resource Has Several Syntaxes

If this is true ...	Use this syntax ...
You want to use the image for *all* clients for which you don't otherwise specify an image. All these clients will have the *same* image.	`Mwm*iconImage:` *path/bitmap*
You want to use the image *only* for a specific class of clients.	`Mwm*`*clientclass*`*iconImage:` *path/bitmap*
You want to use the image *only* for a specific instance of a client named using the client's `name` resource.	`Mwm*`*clientname*`*iconImage:` *path/bitmap*
You want to use the image as the default image whenever the client class or name isn't known.	`Mwm*defaults*iconImage:` *path/bitmap*

For example, if you want to use your own happyface bitmap for **hpterm** windows and see a complete label whenever any icon is active, you would have the following lines in your .Xdefaults file:

```
Mwm*Hpterm*iconImage: /users/yourusername/Bitmaps/face.bits
Mwm*iconDecoration: label activelabel
```

Coloring Icons by Client Class

As it does for window frames, the OSF/Motif Window Manager supplies a number of resources that enable you to specify the colors of icon elements.

Coloring Icon Elements Individually

The following table lists icon image elements and the resources that control their color.

Table 6-13. Coloring Icons with Window Manager Resources

To color this ...	Use this resource ...
Icon image background.	iconImageBackground
Left and upper bevel of icon image.	iconImageTopShadowColor
Right and lower bevel of icon image.	iconImageBottomShadowColor
Icon image foreground.	iconImageForeground

If you do not choose to color an element of an icon image, the window manager uses default values. It gets these values from either of the two following lines:

Mwm*_resource_: _color_ *Colors every instance of an element.*

Mwm*icon*_resource_: _color_ *Colors only this element of icons.*

You can find these lines in the .Xdefaults and sys.Xdefaults files.

When making changes, don't confuse an *element* (foreground, background, topShadowColor) with a *resource* (iconImageForeground, iconImageBackground, iconImageTopShadowColor).

Changing the Tile of Icon Images

The OSF/Motif Window Manager has resources that let you tile the bevels of icon images.

Table 6-14. Tiling Icon Images with Window Manager Resources

To tile this ...	Use this resource ...
Right and lower bevels of an icon image.	`iconImageBottomShadowPixmap`
Left and upper bevels of an icon image.	`iconImageTopShadowPixmap`

Default values for these resources are the icon's bottom and top shadow pixmaps specified using the `bottomShadowPixmap` and `topShandowPixmap` resources set by the entries `Mwm*icon*`*resource* or `Mwm*`*resource*.

The Syntax for Icon Coloring Resources

The resources that color icons can be specified in four ways:

- The first syntax is the most specific. You can use it to specify a resource and value for a particular instance of a client, identified by the client name. (The client name is specified by the client's name resource.) The colors you specify with this resource take precedence over any other specification for this resource.

  `Mwm*`*clientname*`*`*resource*: *color*

- The second syntax specifies a resource and value for a specific client class.

  `Mwm*`*clientclass*`*`*resource*: *color*

- The third syntax specifies a resource and value generally across any and all clients. An example of proper use would be to ensure that all your icon backgrounds were the same color, a good thing for consistency.

  `Mwm*`*resource*: *color*

- The fourth syntax is a default syntax. It specifies the color of any client that is of unknown class.

  `Mwm*default*`*resource*: *color*

Using the Icon Box to Hold Icons

The OSF/Motif Window Manager allows you to use an icon box to contain icons, rather than having stand-alone icons on the workspace. An icon box consists of an **mwm** window and frame. All icons are contained within the icon box. Thus, an icon box can reduce the amount of "real estate" taken up by client icons.

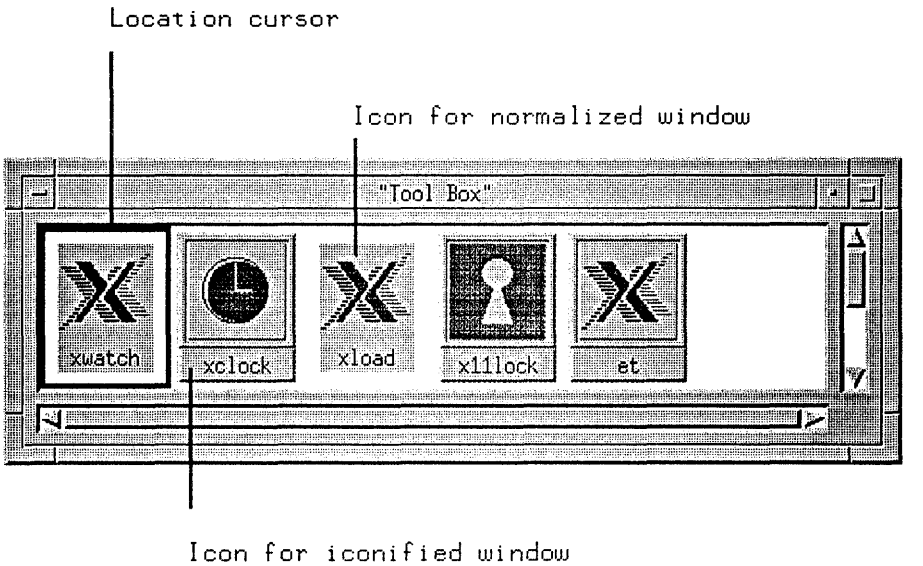


Figure 6-4. Icon Box

The icon box is a scrollable window that displays icons in a grid (rows and columns). Icons in the icon box do not overlap. If there are icons that cannot be displayed in the visible part of the icon box, the user can scroll to see the icons. The sliders within the scroll bars show the extent of the icon grid that is visible.

The icon box can be minimized (iconified) just like any other window. If the icon box is minimized, it is placed into the icon grid on the workspace.

Specifying the Icon Box

Several resources specify whether an icon box is used, define its geometry and location, and specify its name (for looking up resources) and title.

- The `useIconBox` resource specifies whether or not an icon box is used. A value of of "True" places icons in an icon box. The default value of "False" places icons on the root window.
- The `iconBoxGeometry` resource sets the initial size and placement of the icon box. If the `iconBoxGeometry` resource is used, the largest dimension of the size determines if the icons are placed in a row or a column. The default policy is to place icons in a row going from left to right, top to bottom.
- The `iconBoxName` resource specifies the name that is used to look up icon box resources. The default name is "iconbox."
- The `iconBoxTitle` resource specifies the name that is used in the title area of the icon box frame. The default name is "Icons."

For example, the following line specifies that icons will be placed in an icon box:

```
Mwm*useIconBox:  True
```

The value of the `iconBoxGeometry` resource is a standard window geometry string with the following syntax:

- *Width* × *Height* [± *x* ± *y*]

If *x* and *y* are not provided, the icon box is placed at +0−0.

The actual size of the icon box window depends on the `iconImageMaximum` (size) and `iconDecoration` resources. The default value for size is (6 * iconWidth + padding) wide by (1 * iconHeight + padding) high.

Controlling the Appearance of Icon Boxes

The icon box is displayed in a standard window management client frame. Client-specific resources for the icon box can be specified using "iconbox" as the client name.

```
Mwm*iconbox*resource:  value
```

Resources that can be used with the icon box to change its appearance are:

- `iconDecoration`.
- The `mwm` resources dealing with mattes and icon appearance. (The icon appearance resources affect the icon displayed when the icon box is minimized.)

The Icon Box Window Menu

The window menu for the icon box differs from the standard window menu in that it does not contain the "Close" selection. In its place is the "PackIcons" selection, which shifts icons to fill empty spaces in the icon placement grid so that the icons appear in neat, complete rows.

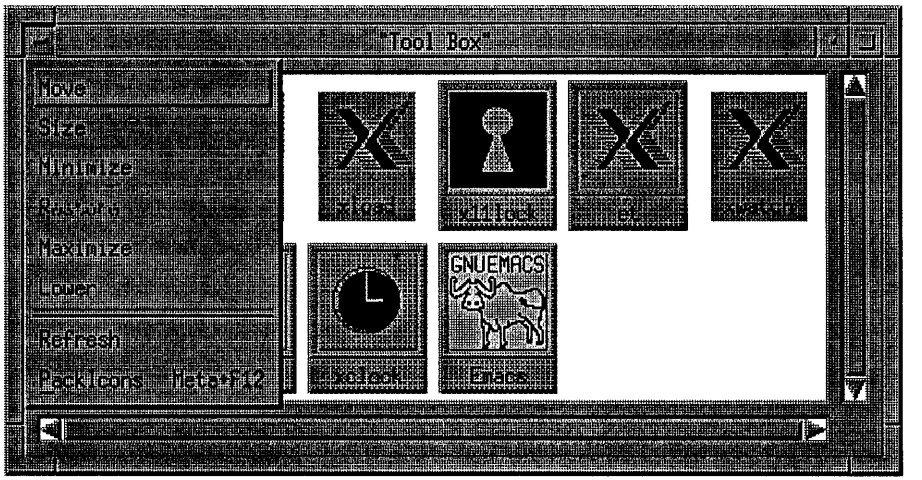


Figure 6-5. Icon Box With Icon Box Window Menu

Controlling Icons in the Icon Box

Every client window that can be iconified has an icon in the icon box, even when the window is in the normal state. The icon for a client is put into the icon box when the client becomes managed by the window manager, and is removed from the icon box when the client withdraws from being managed.

Icons for windows in the normal (open) state are visually distinct from icons for windows that are iconified. Icons for windows that are iconified look like stand-alone icons. Icons for windows that are in the normal state appear flat and are optionally grayed-out. The value of "True" for the `fadeNormalIcon` resource grays out icons for normalized windows. The default value is "False."

The text and image attributes of icons in icon boxes are determined in the same way as for stand-alone icons, using the `iconDecoration` resource.

A standard "control" location cursor is used to indicate the particular icon in the icon box to which keyboard actions apply. The location cursor is an unfilled rectangle that surrounds the icon.

Icons contained in the icon box can be manipulated with the mouse and from the keyboard. Mouse button actions apply whenever the pointer is on any part of the icon.

Table 6-15. Controlling Icons in the Icon Box With a Mouse

If you want to ...	Do this ...
Select an icon.	Press button 1.
Normalize (open) an iconified window.	Double-click mouse button 1.
Raise a normalized window to the top of the stack.	Double-click mouse button 1.
Move an icon within the icon box.	Drag button 1.

To manipulate an icon from the keyboard, make the icon box the active window and use the arrow keys to traverse the icons in the icon box. Pressing Return does the default action for the selected icon: for an icon of a normalized window, the window is raised; for an icon of an iconified window, the window is normalized. The arrow keys move the focus around the icons that are visible. The Tab key moves the keyboard input focus around the box in this order: icons, horizontal scroll bar, vertical scroll bar, icons. Shift Tab moves the focus in the opposite direction.

Managing Window Manager Menus

Menus offer an easy way to get the system to do something for you. While the concepts of "operating system," "commands," and "arguments" are confusing to the inexperienced user, most people can readily appreciate the concept of choosing a selection from a menu.

The OSF/Motif Window Manager menus are controlled by a text file in the `/usr/lib/X11` directory called `system.mwmrc`, unless you have a file in your home directory called `.mwmrc`. You can add or delete menus and menu selections by copying `system.mwmrc` to your home directory as `.mwmrc` and modifying it to suit your needs.

Default Menus

The OSF/Motif Window Manager comes with two default menus:

- The Window Menu.
- The Root Menu.

The default window menu is built into `mwm`. For reference, a copy of its contents are placed in `.mwmrc`.

```
Menu DefaultWindowMenu
{
    "Restore"    _R      Alt<Key>F5      f.normalize
    "Move"       _M      Alt<Key>F7      f.move
    "Size"       _S      Alt<Key>F8      f.resize
    "Minimize"   _n      Alt<Key>F9      f.minimize
    "Maximize"   _x      Alt<Key>F10     f.maximize
    "Lower"      _L      Alt<Key>F3      f.lower
    no-label                             f.separator
    "Close"      _C      Alt<Key>F4      f.kill
}
```

Table 6-16. Action of Entries in the Window Menu

Entry	Action Taken
Menu DefaultWindowMenu	Function type and name for window menu.
Restore	Normalizes icon or maximized window.
Move	Moves window around screen.
Size	Changes window size.
Minimize	Changes window into icon.
Maximize	Enlarges window to cover screen.
Lower	Lowers window to bottom of stack.
Close	Closes window by killing its process.

The `windowMenu` resource must be set in order to replace the `DefaultWindowMenu` with a different menu.

The default root menu is specified in the same files by the following lines:

```
Menu RootMenu
{
 "Root Menu"     f.title
 "New Window"    f.exec "hpterm &"
 "Start Clock"   f.exec "xclock -geometry 100x90-1+1 &"
 "Start Load"    f.exec "xload -geometry 150x90-130+1 &"
 "Shuffle Up"    f.circle_up
 "Shuffle Down" f.circle_down
 "Refresh"       f.refresh
  no-label       f.separator
 "Restart ... "   f.restart
}
```

By default, the window menu displays when you do the following operations:

- Press button 1 on a window frame's window menu button.
- Press button 3 anywhere on a window frame.
- Press Shift Esc with the keyboard focus set to a window.

By default, the root menu displays when you press button 3 on the root window.

You can modify either menu to suit the specific needs of your application; however, for the sake of the consistency of window operation, it's usually better to modify the root menu and keep the window menu the same.

Modifying Menu Selections and Their Functions

All window manager menus, regardless of the mechanism that calls them to the screen, have the same syntax.

Menu Syntax

```
Menu MenuName
{
    selection1 [mnemonic]   [accelerator]   function [argument]
    selection2 [mnemonic]   [accelerator]   function [argument]
    selection3 [mnemonic]   [accelerator]   function [argument]
                                    ⋮
    selection* [mnemonic]   [accelerator]   function [argument]
}
```

Each line identifies a selection name followed by the function to be done if that selection is chosen. The order of the selections is the order of their appearance when you display the menu. A selection name may be either a character string or a bitmap.

Modifying Selections

Any character string containing a space must be enclosed in double quotes (""); single-word strings don't have to be enclosed, but it's probably a good idea for the sake of consistency. An alternate method of dealing with two-word selection names is to use an underbar (_) in place of the space.

You can create a bitmap with the `bitmap` client and use it as a selection name. The syntax for doing this is as follows:

@/path/bitmapfile function [*argument*]

Note the at-sign (@) in the above line. The at-sign tells the window manager that what follows is the path to a bitmap file.

Modifying Functions

Each function operates in one or more of the following contexts:

root	Operates the function when the root window is selected.
window	Operates the function when a client window is selected. Some functions operate only when the window is in its normalized state (f.maximize), or its maximized or iconified state (f.normalize).
icon	Operates the function when an icon is selected.

Additionally each function is operated by one or more of the following devices:

- Button.
- Key.
- Menu.

Any selection that uses an invalid context, an invalid function, or a function that doesn't apply to the current context is grayed out. This is the case with the "Restore" selection of a terminal window's system menu or the "Minimize" selection of an icon's window menu.

The following table lists the valid functions for the OSF/Motif Window Manager.

Table 6-17. Valid Window Manager Functions

Functions		Contexts			Devices		
Name	**Description**	**Root**	**Icon**	**Window**	**Button**	**Key**	**Menu**
f.beep	Causes a beep to sound.	√	√	√	√	√	√
f.circle_down	Puts window on bottom of stack.	√	√	√	√	√	√
f.circle_up	Puts window on top of stack.	√	√	√	√	√	√
f.exec	Uses `/bin/sh` to execute a command.	√	√	√	√	√	√
f.focus_color	Sets colormap focus when colormap focus policy is explicit.	√	√	√	√	√	√
f.focus_key	Sets keyboard input focus when keyboard focus policy is explicit.	√	√	√	√	√	√
f.kill	Terminates a client's connection to server.		√	√	√	√	√
f.lower	Lowers a window to bottom of stack.		√	√	√	√	√
f.maximize	Enlarges a window to its maximum size.		√	√	√	√	√
f.menu	Associates a menu with a selection or binding.	√	√	√	√	√	√
f.minimize	Changes a window into an icon.			√	√	√	√
f.move	Enables the interactive moving of a window.		√	√	√	√	√

Table 6-17a. Valid Window Manager Functions (continued)

Functions		Contexts			Devices		
Name	**Description**	**Root**	**Icon**	**Window**	**Button**	**Key**	**Menu**
f.next_cmap	Installs the next colormap in the window with the colormap focus.	√	√	√	√	√	√
f.next_key	Sets keyboard focus policy to the next window/icon in the stack.	√	√	√	√	√	√
f.nop	Does no function.	√	√	√	√	√	√
f.normalize	Displays a window in normal size.		√	√	√	√	√
f.pack_icons	Packs icons rows in the root window or icon box.	√	√	√	√	√	√
f.pass_keys	Toggles between enabling and disabling processing of key bindings.	√	√	√	√	√	√
f.post_wmenu	Posts the window menu	√	√	√	√	√	
f.prev_cmap	Installs the previous color map in the window with the colormap focus.	√	√	√	√	√	√
f.prev_key	Sets the keyboard input focus to the next window/icon in the stack.	√	√	√	√	√	√

Table 6-17b. Valid Window Manager Functions (continued)

Functions		Contexts			Devices		
Name	**Description**	**Root**	**Icon**	**Window**	**Button**	**Key**	**Menu**
f.quit_mwm	Terminates OSF/Motif Window Manager, but not X.	√			√	√	√
f.raise	Lifts a window to the top of the window stack.		√	√	√	√	√
f.raise_lower	Raises a partially concealed window; lowers an unconcealed window.		√	√	√	√	√
f.refresh	Redraws all windows.	√	√	√	√	√	√
f.refresh_win	Redraws a client window.			√	√	√	√
f.resize	Enables you to interactively resize a window.			√	√	√	√
f.restart	Restarts the OSF/Motif Window Manager.	√			√	√	√
f.send_msg	Sends a client message.		√	√	√	√	√
f.separator	Draws a line between menu selections.	√	√	√			√
f.set_behavior	Restarts `mwm` with CXI or custom behavior.	√	√	√	√	√	√
f.title	Inserts a title into a menu at the specified position.	√	√	√			√

Menu Titles

The `f.title` functions creates a menu title, and automatically places a separator above and below the title.

Menu Selections

A *selection* can be a character string or a graphic bitmap. If the character string contains spaces, it must be enclosed in quotation marks (").

Mnemonics and Accelerators

You have the option of using a mnemonic and accelerator with a menu selection. A mnemonic is specified using the syntax:

`mnemonic = _`*character*

An accelerator is specified using keyboard binding syntax described later in this chapter (see "Keyboard Binding Syntax").

< idx|menus:changing|

Changing the Menu Associated with the Window Menu Button

The `windowMenu` resource lets you change the *menu* displayed when you press button 1 on the window menu button. This gives you the ability to display a menu of your choice from the window menu button. All you need do is make a new menu, then use the `windowMenu` resource to associate this new menu with the window menu button.

The `windowMenu` resource has several forms.

- The first syntax specifies a menu for a particular instance of a client, identified by the client name. (The client name is specified by the client's `name` resource.)

  `Mwm*`*clientname*`*windowMenu:` *MenuName*

  For example, you would place the following line in your `.Xdefaults` file to associate a menu named `EditMenu` with an `hpterm` window started as `hpterm -name hp850`.

  ```
  Mwm*hp850*windowMenu: EditMenu
  ```

- The second syntax specifies the menu for a specific class of clients:

  `Mwm*`*`clientclass`*`*windowMenu:` *MenuName*

  For example, you may want to associate a particular `EditorMenu` of your own creation with all `hpterm` windows:

  ```
  Mwm*HPterm*windowMenu:  EditorMenu
  ```

- The third syntax specifies the menu for all classes of clients:

  `Mwm*windowMenu:` *MenuName*

  For example, if you want to associate a special `CADCAMMenu` menu with the window menu button, you would add the following line to your `.Xdefaults` file:

  ```
  Mwm*windowMenu: CADCAMMenu
  ```

- The fourth syntax specifies a menu for any client whose class is unknown:

  `Mwm*defaults*windowMenu:` *MenuName*

Making New Menus

You have the option of modifying the window and root menus, but you also have another option: You can create a completely new menu. The new menu can be displayed by a button press, a key press, or by selecting it from an existing menu.

To create a completely new menu, use the menu syntax in the previous section as a model to do the following:

1. Fill in a menu name.
2. Make up selection names.
3. Optional: Choose a mnemonic and accelerator.
4. Give each selection a function to perform from the "Table of Menu Functions."

For example, the following menu is named "Graphics Projects." The menu selections are all bitmaps symbolizing different graphics projects. The bitmaps

are kept in this user's home directory **/users/dub**. When the user, Dub, selects a symbol, the graphics program starts and opens the appropriate graphics file.

```
Menu "Graphics Projects"
{
@/users/dub/fusel.bits _F Alt<Key>F f.exec "cad /spacestar/fusel.e12"
@/users/dub/lwing.bits _L Alt<Key>L f.exec "cad /spacestar/lwing.s05"
@/users/dub/rwing.bits _R Alt<Key>R f.exec "cad /spacestar/rwing.s04"
@/users/dub/nose.bits  _N Alt<Key>N f.exec "cad /spacestar/nose.e17"
}
```

To display a new menu with a button or key, follow these steps:

1. Choose the button or key that you want to use.
2. Choose the action on the button or key that will cause the menu to display.
3. Use the **f.menu** function with the menu name as an argument to bind the menu to the button or key.

For more information on button and keyboard bindings, including examples, see the next two sections, "Using the Mouse" and "Using the Keyboard."

Using the Mouse

The mouse offers a quick way to make things happen in your window environment without the time-consuming process of typing commands. The window manager recognizes the following button operations:

Press	Holding down a mouse button.
Click	Pressing and releasing a mouse button.
Double-click	Pressing and releasing a mouse button twice in rapid succession.
Drag	Pressing a mouse button and moving the pointer (and mouse device).

You associate a button operation with a window management function using a **button binding.** A button binding is a command line you put in the

Default Button Bindings

The OSF/Motif Window Manager comes with the following built-in button bindings.

Table 6-18. Built-In Button Bindings

Location of Pointer	Behavior
Window menu button	Pressing button 1 displays the window menu. This behavior can be modified by the **wMenuButtonClick** resource.
Window menu button	Double-clicking button 1 closes the window. This behavior can be modified by the **wMenuButtonClick2** resource.
Minimize button	Clicking button 1 minimizes the window.
Maximize button	Clicking button 1 maximizes the window.
Title bar	Dragging button 1 moves the window.
Window or icon	Pressing button 1 gives it keyboard focus.
Resize border	Dragging button 1 resizes the window.
Icon	Clicking button 1 displays the icon window menu. This behavior can be modified by the **iconClick** resource.
Icon	Double-clicking button 1 normalizes the window.
Icon	Pressing button 1 moves the icon.

These bindings are fixed—they cannot be *replaced* by other bindings. However, you can *add* to some of them (see "Modifying Button Bindings and Their Functions.") For example, you can specify an additional function for double-clicking button 1 in an icon, but the double click will also normalize the window.

Mwm provides an additional default binding that can be deleted or replaced:

Table 6-19. Additional Button Bindings

Locaton of Pointer	Behavior
Icon or Frame	Pressing button 1 raises the window or icon.

This binding is listed in the following section of the `.mwmrc` file.

```
Buttons DefaultButtonBindings
{
 <Btn1Down>  icon|frame  f.raise
}
```

The binding can be removed or altered by deleting or editing the line that begins with `<Btn1Down>`. (In order for the editing to have an effect, the `buttonBindings` resource in the `.Xdefaults` file must be set to `DefaultButtonBindings`, and you must restart the window manager.)

Modifying Button Bindings and Their Functions

You can modify the button bindings section of your `.mwmrc` file to suit your individual needs.

Button Binding Syntax

The syntax for button bindings is as follows:

```
Buttons ButtonBindingSetName
{
    button   context|context   function   [argument]
    button   context|context   function   [argument]
    button   context|context   function   [argument]
}
```

Each line identifies a certain button and operation, followed by the context in which the button operation is valid, followed by the function to be done. The following button binding contexts are recognized by the window manager:

root — Operates the function when the button is activated in the root window.

window	Operates the function when the button is activated in a client window or window frame.
frame	Operates the function when the button is activated on a window frame.
icon	Operates the function when the button is activated on an icon.
title	Operates the function when the button is activated on a title bar.
app	Operates the function when the button is activated in a client window (excludes window frames).

Modifying Button Bindings

To modify the button bindings:

1. Edit the "DefaultButtonBindings" section of the `system.mwmrc` (to make system-wide changes) or `.mwmrc` (to make changes to your local environment). The easiest way to modify button bindings is to change the default binding or to insert extra lines.
2. Set the `buttonBindings` resource in your `.Xdefaults` file to `DefaultButtonBindings`. This tells `mwm` to look in the "DefaultButtonBindings" section of your `.mwmrc` file for a list of button bindings.

For example, Dub, the user who created his own "Graphics Project" menu in the previous section, may want to display the menu when he presses the Extend char–button 3 sequence with the pointer in the root window. He would only need to insert one line in his `.mwmrc` file to make this happen. The "DefaultButtonBindings" section of his `.mwmrc` file would look like the following:

```
Buttons DefaultButtonBindings
{
 <Btn1Down>      icon|frame  f.raise
 Alt<Btn3Down>   root        f.menu  "Graphics Project"
}
```

In addition, Dub must have the following entry in his `.Xdefaults` file:

```
Mwm*buttonBindings:  DefaultButtonBindings
```

The new key binding will be in effect when Dub restarts the window manager.

Making a New Button Binding Set

Perhaps inserting a new button binding into the "DefaultButtonBindings" set is not enough. Perhaps you need to make a complete new set of button bindings. To do this, use the "DefaultButtonBindings" section of your `.mwmrc` as a model. After you have created the new button binding set, use the `buttonBindings` resource to tell the window manager about it. You do this by adding a line to your `.Xdefaults` file. The syntax of the line is as follows:

```
Mwm*buttonBindings: NewButtonBindingSet
```

This line directs the window manager to use "NewButtonBindingSet" as the source of its button binding information. The button bindings are assumed to exist in the file named by the `Mwm*configFile:` resource, the default being `.mwmrc`.

For example, suppose Dub, our graphics user, wants to specify a completely new button binding set instead of inserting a line in the existing "DefaultButtonBindings" set. He needs to create a new button binding set, such as the following, modeled after the default set:

```
Buttons GraphicsButtonBindings
{
 <Btn2Down>  root  f.menu  "Graphics Project"
}
```

In his `.Xdefaults` file, Dub would then insert the following line:

```
Mwm*buttonBindings: GraphicsButtonBindings
```

To display his graphics menu, Dub needs only to press button 2 when the pointer is on the root window.

Modifying Button Click Timing

The OSF/Motif Window Manager has another resource for controlling button behavior. This resource, `doubleClickTime`, sets the maximum time (in milliseconds) that can elapse between button clicks before a double-click becomes just "two clicks in a row." In other words, if two clicks occur in less than the maximum time, they are assumed to be a double-click; if two clicks

occur in a time greater than the maximum time, they are assumed to be two single clicks. The default is 500 (milliseconds).

Using the Keyboard

Similar to mouse button bindings, you can bind (associate) window manager functions to "special" keys on the keyboard using **keyboard bindings.** The window manager recognizes the following special keys:

- Shift.
- Escape.
- Alt (Meta or Extend Char).
- Tab.
- Ctrl.
- Lock.

Default Key Bindings

The OSF/Motif Window Manager comes with the following default key bindings.

Table 6-20.
OSF/Motif Window Manager Default Keyboard Bindings

When the keyboard focus is:	Press:	What this does is:
Window or icon	Shift Escape	Displays window menu.
Window or icon	Alt space	Displays window menu.
Window, icon, or none	Alt Tab	Switches keyboard focus to the next window or icon.
Window, icon, or none	Alt Shift Tab	Switches keyboard focus to the previous window or icon.
Window, icon, or none	Alt Escape	Switches keyboard focus to the next window or icon.
Window, icon, or none	Alt Shift Escape	Switches keyboard focus to the previous window or icon.
Window	Alt F6	Switches keyboard focus to the next window or icon, including transient windows.
Window, icon, or none	Alt Ctrl Shift !	Restart `mwm` with default or custom behavior.

These keyboard bindings are listed in the following lines in `system.mwmrc` and `.mwmrc`.

```
Keys DefaultKeyBindings
{
    Shift<Key>Escape           window          f.post_wmenu
    Alt<Key>space              window|icon     f.post_wmenu
    Alt<Key>Tab                root|icon|window f.next_key
    Alt Shift<Key>Tab          root|icon|window f.prev_key
    Alt<Key>Escape             root|icon|window f.next_key
    Alt Shift<Key>Escape       root|icon|window f.prev.key
    Alt<Key>F6                 window          f.next_key transient
    Alt Ctrl Shift<Key>exclam root|icon|window f.set_behavior
```

The first line specifies the function type (`Keys`) and the name of the keyboard binding set (`DefaultKeyBindings`). The following lines specify the actual keyboard bindings. For example, the first line binds the Shift Esc key press sequence to the function that displays the window menu. The second line binds the Extend char space key press sequence to the same function.

You can modify or delete any of these bindings by editing or deleting the line. (In order for the editing to have an effect, the `keyBindings` resource in the `.Xdefaults` file must be set to `DefaultKeyBindings`.)

Modifying Keyboard Bindings and Their Functions

You can modify the keyboard bindings section of your `.mwmrc` file if your situation requires it.

Keyboard Binding Syntax

The syntax for keyboard bindings is as follows:

```
Keys KeyBindingSetName
{
    key   context|context   function    [argument]
    key   context|context   function    [argument]
    key   context|context   function    [argument]
}
```

Each line identifies a unique key press sequence, followed by the context in which that sequence is valid, followed by the function to be done. The following keyboard binding contexts are recognized by the window manager:

root	Operates the function when the key is pressed while the root window has keyboard focus.
window	Operates the function when the key is pressed while a client window has keyboard focus.
icon	Operates the function when the key is pressed while an icon has keyboard focus.

Modifying Keyboard Bindings

To modify the default keyboard bindings bindings:

1. Edit the "DefaultKeyBindings" section of the `system.mwmrc` (to make system-wide changes) or `.mwmrc` (to make changes to your local environment). The easiest way to modify button bindings is to change the default binding or to insert extra lines.
2. Set the `keyBindings` resource in your `.Xdefaults` file to `DefaultKeyBindings`. This tells `mwm` to look in the "DefaultKeyBindings" section of your `.mwmrc` file for a list of key bindings.

For example, suppose Dub, the user who created his own "Graphics Project" menu, kept pressing the Shift Esc sequence and accidentally displaying the window menu. He might decide that he is better off to disable that particular keyboard binding. To do so, he would need to delete (or comment out) the proper line in his `.mwmrc` file:

```
#   Shift<Key>Escape      window        f.post_wmenu
```

Dub commented out the line by placing a hash mark (#) in the left margin of the line.

In addition, Dub must have the following entry in his `.Xdefaults` file:

```
Mwm*keyBindings:  DefaultKeyBindings
```

Making a New Keyboard Binding Set

With keyboard bindings, as with button bindings, you have the option of creating a whole new binding set. To do so, you can use the "DefaultKeyBindings" section of your `.mwmrc` as a model. After you have created the new keyboard binding set, use the `keyBindings` resource to explain your modification to the window manager. You do this by adding a line to your `.Xdefaults` file. The syntax of the line is as follows:

`Mwm*keyBindings:` *NewKeyboardBindingSet*

This line directs the window manager to get its keyboard binding information from the "NewKeyboardBindingSet" section of the `.hpwmrc` file. You can have the window manager look in any file if you specify the path and file name with the `Hpwm*configFile:` resource in your `.Xdefaults` file.

Customizing the Windows Frames

In some cases, you might feel your "screen real estate" is too expensive, and you may not want to take up "valuable space" with window frames. For example, do you really need functional buttons and a resizable frame around a clock that just sits in the corner of your screen? You could switch to the **uwm** window manager, but the OSF/Motif Window Manager has two resources for just such situations:

- The `clientDecoration` resource enables you to choose just how much or how little "decoration" you want to put around each client.
- The `transientDecoration` resource enables you to choose just how much or how little decoration you want to put around each transient window. A **transient window** is a relatively short-lived window, for example, a dialog box.

You can still access the functionality of any decoration you remove by binding its functions to mouse buttons or to key presses, as explained in the above sections.

Adding or Removing Frame Elements

You specify the `clientDecoration` and `transientDecoration` resources as a list of the frame elements. If the first element in the list is preceded by a plus (+) sign or has no sign preceding it, the window manager starts with no frame and assumes that the list contains those elements you want added. If the list begins with a minus (−) sign, the window manager starts with a complete frame and assumes that the list contains elements you want removed from the frame.

The list of valid window frame elements is as follows:

Table 6-21.
Valid OSF/Motif Window Manager Window Frame Elements

Frame Element	Description
all	Include all frame elements (default value).
border	Window border.
maximize	Maximize button (includes title bar).
minimize	Minimize button (includes title bar).
none	Include no window frame elements.
resizeh	Resize border handles (includes border).
menu	Window menu button (includes title bar).
title	Title bar (includes border).

The Syntax for the 'clientDecoration' and 'transientDecoration' Resources

The `clientDecoration` resource has several forms.

- The first syntax the most specific. It specifies the addition or removal of elements from a particular instance of a client, identified by the client name. (The client name is specified by the client's `name` resource.)

`Mwm*`*clientname*`*clientDecoration:` [{ ±all | ±none } | ±border | ±maximize | ±minimize | ±resizeh | ±menu | ±title]

For example, you may want a border with only a title bar and window menu button around a particular `hpterm` window started as `hpterm -name hp850`.

```
Mwm*hp850*clientDecoration: +menu
```

- The second syntax specifies the addition or removal of elements from specific classes of clients:

```
Mwm*clientclass*clientDecoration: [ { ±all } ]
                                  [ { ±none} ]
                                  [ ±border  ]
                                  [ ±maximize]
                                  [ ±minimize]
                                  [ ±resizeh ]
                                  [ ±menu    ]
                                  [ ±title   ]
```

  For example, you may want to remove just the resize border and maximize button from all clocks you display on your screen:

```
Mwm*XClock.clientDecoration: -resizeh -maximize
```

- The third syntax specifies the addition or removal of elements from all classes of clients:

```
Mwm*clientDecoration: [ { ±all } ]
                      [ { ±none} ]
                      [ ±border  ]
                      [ ±maximize]
                      [ ±minimize]
                      [ ±resizeh ]
                      [ ±menu    ]
                      [ ±title   ]
```

  For example, you could remove the maximize button from all windows by adding the following line in your `.Xdefaults` file:

```
Mwm*clientDecoration: -maximize
```

- The fourth syntax specifies the addition or removal of elements from any client with an unknown class:

```
Mwm*defaults*clientDecoration: [ { ±all } ]
                               [ { ±none} ]
                               [ ±border  ]
                               [ ±maximize]
                               [ ±minimize]
                               [ ±resizeh ]
                               [ ±menu    ]
                               [ ±title   ]
```

The `transientDecoration` resource has the following syntax:

```
Mwm*transientDecoration: [ { ±all } ]
                         [ { ±none} ]
                         [ ±border  ]
                         [ ±maximize]
                         [ ±minimize]
                         [ ±resize  ]
                         [ ±menu    ]
                         [ ±title   ]
```

For example, you could remove the tile bar from all transient windows by adding the following line in your `.Xdefaults` file:

```
Mwm*transientDecoration: -title
```

Controlling Window Size and Placement

The OSF/Motif Window Manager has several resources that allow you to refine your control of the size and placement of windows.

Refining Control with Window Manager Resources

The following table lists window manager resources enabling you to refine your control over the size and placement of windows.

Table 6-22.
Refining Your Control with Window Manager Resources

To control this ...	Use this resource ...	The default is ...
Initial placement of new windows on the screen.	`interactivePlacement`	False
The ability to enlarge windows beyond the size specified in `maximumClientSize`.	`limitResize`	False
The maximum size of a client window set by user or client.	`maximumMaximumSize`	2×screen
The sensitivity of dragging operations.	`moveThreshold`	4 pixels
Exact positioning of window and window frame.	`positionIsFrame`	True
Clipping of new windows by screen edges.	`positionOnScreen`	True
The width of the resize border of the window frame.	`resizeBorderWidth`	10 pixels
Displaying the resize cursors when the pointer is in the resize border.	`resizeCursors`	True
The maximum size of a maximized client.	`maximumClientSize`	screen size

The `interactivePlacement` resource has the following values:

True	The pointer changes shape (to an upper left corner bracket) before a new window displays, so you can choose a position for the window.
False	The pointer doesn't change shape. A new window displays according to the placement values specified in the X configuration files.

The `limitResize` resource has the following values:

True	A window cannot be resized to greater than the maximum size specified by the `maximumClientSize` resource or the `WM_NORMAL_HINTS` window property.
False	A window can be resized to any size.

The value of the `maximumMaximumSize` resource is the width×height of the screen being used. The dimensions are given in pixels. For example, for an SRX display, `maximumMaximumSize` would have a value of 1280×1024.

The value of the `moveThreshold` resource is the number of pixels that the pointer must be moved with a button pressed before a move operation is initiated. You can use this resource to prevent window or icon movement when you unintentionally move the pointer during a click or double-click.

The `positionIsFrame` resource has the following values:

True	The position information (from `WM_NORMAL_HINTS` and configuration files) refers to the position of the window frame.
False	The position information refers to the position of the window itself.

The `positionOnScreen` resource has the following values:

True	If possible, a window is placed so that it is not clipped. If not possible, a window is placed so that at least the upper left corner of the window is on the screen.
False	A window is placed at the requested position even if it is totally off the screen.

The value of the **resizeBorderWidth** resource is the width of the resize border, the outermost portion of the window frame. The width is measured in pixels.

The **resizeCursors** resource has the following values:

True	The appropriate resize cursor displays when the pointer enters a resize border area of the window frame.
False	The resize cursors are not displayed.

The value of the **maximumClientSize** resource is the width×height (in pixels) of the maximum size of a maximized client. If this resource isn't specified, the maximum size is taken from the **WM-NORMAL_HINTS** window property, or the default size (the size of the screen) is used.

The Syntax for Size and Position Refinement Resources

The resources that refine your control over the size and placement of windows have the following syntax:

```
Mwm*resource: value
```

For example, you could choose to place each new window on the screen interactively by adding the following line in your **.Xdefaults** file:

```
Mwm*interactivePlacement: True
```

In addition to this syntax, the **maximumClientSize** resource has two more forms.

- Use this syntax for specifying the maximum client size for a specific instance of a client:

  ```
  Mwm*clientname.maximumClientSize: width×height
  ```

- Use this syntax for specifying the maximum client size for specific classes of clients:

  ```
  Mwm*clientclass.maximumClientSize: width×height
  ```

  For example, you might decide that **xload** clients should be maximized to no more than an eighth of the size of your 1024×768 display.

  ```
  Mwm*XLoad.maximumClientSize: 128×96
  ```

- Use this syntax for specifying the maximum client size for any client with an unknown class.

  `Mwm*defaults*maximumClientSize:` *width*`x`*height*

Controlling Resources with Focus Policies

The OSF/Motif Window Manager has three separate focus policies that you can use to control the arbitration of resources among clients. The focus policies determine what happens when a window becomes the active window. The active window is the window that has the focus of the keyboard and any extended input devices. When a window is active, the following are true:

- What you type appears in that window.
- The color of the window frame changes to indicate the active focus.
- Input from extended input devices goes to that window.

Each focus policy is controlled by a specific focus policy resource. The focus policy resources are as follows:

Table 6-23.
Controlling Focus Policies with Window Manager Resources

To control this ...	Use this resource ...	The default value is ...
Which client window has the colormap focus.	`colormapFocusPolicy`	keyboard
Which client window has the keyboard and mouse focus.	`keyboardFocusPolicy`	explicit

Valid Focus Policies

The following focus policies are valid for the `colormapFocusPolicy` resource:

keyboard — The window manager tracks keyboard input and installs a client's colormap when the client window gets the keyboard input focus.

pointer — The window manager tracks the pointer and installs a client's colormap when the pointer moves into the client window or the window frame around the client.

explicit — The window manager tracks a specific focus-selection operation and installs a client's color map when the focus-selection operation is done in the client window.

The following focus policies are valid for the `keyboardFocusPolicy` resource:

pointer — The window manager tracks the pointer and sets the keyboard focus to a client window when the pointer moves into that window or the window frame around the client.

explicit — The window manager tracks a specific focus-selection operation and sets the keyboard focus to a client window when the focus-selection operation is done in that client window.

When the keyboard focus policy is explicit, you can use the `passSelectButton` resource to specify the consequence of the focus-selection operation. If you give `passSelectButton` a value of "True" (the default value), the focus-selection operation is passed to the client or used by the window manager to perform some action. If you give `passSelectButton` a value of "False," the focus-selection operation will be used only to select the focus and will not be passed.

The Syntax of Focus Policy Resources

The focus policy resources have the following syntax:

`Mwm*`*`focusPolicyResource`*`:` *policy*

For example, you could change the keyboard focus policy so that moving the pointer into a window moved the focus there by adding the following line in your `.Xdefaults` file:

```
Mwm*keyboardFocusPolicy: pointer
```

Matting Clients

If you have a color system, you might find it useful to decorate windows based on client class or client name. For example, you may wish to color code your `hpterm` windows so you can easily tell them apart from your `xterm` windows.

You can accomplish this differentiation by using a **matte** and the window manager's matte resources to further frame your client windows. A matte is a 3-D border just inside the window between client area and window frame.

To enable a matte, define a positive matte width for windows of a specific class. Careful selection of matte elements will give a pleasing 3-D effect.

To define a matte width, use the `matteWidth` resource. The `matteWidth` resource defines the width of the matte or border between the client and the window frame. The width is given in pixels. For example, to specify a matte of 10 pixels around `hpterm` windows, you would include the following line in your `.Xdefaults` file:

```
Mwm*HPterm.matteWidth:  10
```

Coloring Individual Matte Elements

The following table lists matte elements and the resources that control their color.

Table 6-24.
Coloring Window Frames with Window Manager Resources

To color this ...	Use this resource ...	The default value is ...
Matte background.	`matteBackground`	`mwm` background
Left and upper bevel of matte.	`matteTopShadowColor`	Lightened `matteBackground` color.
Right and lower bevel of matte.	`matteBottomShadowColor`	Darkened `matteBackground` color.
Matte foreground.	`matteForeground`	Darkened `matteBottonShadowColor`.

Changing the Tile of Mattes

As with frame colors, the fewer the number of colors your display can produce, the more interest you will probably have in tiling mattes. Again, tiling provides you with a way to "mix" foreground and background colors into a third color. If you have a 2-color (monochrome) display, you could tile a window matte in shades of gray to achieve a pleasing 3-D look.

The OSF/Motif Window Manager has the following resources to enable you to tile mattes.

Table 6-25. Tiling Mattes with Window Manager Resources

To tile this ...	Use this resource ...	The default value is ...
Matte right and lower bevels.	`matteBottomShadowPixmap`	client bottom shadow color
Matte left and upper bevels.	`matteTopShadowPixmap`	client top shadow color

The following table lists the acceptable values for pixmap resources:

Table 6-26. The Values to Use for Tiling Mattes

To tile an element this color mix ...	Use this value ...
The foreground color.	foreground
The background color.	background
A mix of 25% foreground to 75% background.	25_foreground
A mix of 50% foreground to 50% background.	50_foreground
A mix of 75% foreground to 25% background.	75_foreground
In horizontal lines alternating between the foreground and background colors.	horizontal_tile
In vertical lines alternating between the foreground and background colors.	vertical_tile
In diagonal lines slanting to the right alternating between the foreground and background colors.	slant_right
In diagonal lines slanting to the left alternating between the foreground and background colors.	slant_left

The Syntax for Matte Resources

Matte resources can have any of the following three syntaxes, depending on the situation:

- The first syntax is the most specific. It creates a matte for a particular instance of a client, identified by the client name. (The client name is specified by the client's `name` resource.)

  `Mwm*`*clientname*`*`*matteResource*`:` *value*

  For example, you could create a 5-pixel-wide LightBlue matte for a particular `hpterm` window started as `hpterm -name hp850`.

  ```
  Mwm*hp850*matteWidth:          10
  ```

- The second syntax specifies the matte for specific classes of clients:

  `Mwm*`*clientclass*`*`*matteResource*`:` *value*

For example, you could place a different matte around `hpterm` and `xterm` windows by including the following lines in your `.Xdefaults` file:

```
Mwm*HPterm*matteWidth:          10
Mwm*HPterm*matteBackground:     SkyBlue
Mwm*XTerm*matteWidth:           10
Mwm*XTerm*matteBackground:      Tan
```

- The third syntax creates a matte for all clients regardless of class:

  `Mwm*`*`matteResource`*`:` *value*

  For example, you could create a 10-pixel-wide yellow matte for every client window by adding the following lines in your `.Xdefaults` file:

```
Mwm*matteWidth:          10
Mwm*matteBackground:     Yellow
Mwm*makeMatteColors:     all
```

- The fourth syntax specifies the matte for any client with an unknown class.

  `Mwm*defaults*`*`matteResource`*`:` *value*

Switching Between Default and Custom Behavior

The window manager has a buit-in key binding that allows you to switch back and forth between customized `mwm` behavior and default behavior. The key presses for doing this are Alt Shift Ctrl !.

The following client-specific resources are affected by this function:

`clientDecoration` `clientFunctions` `focusAutoRaise` `windowMenu`

The following `mwm` resources are affected by this function:

autoKeyFocus	buttonBindings	clientAutoPlace
colormapFocusPolicy	configFile	deiconifyKeyFocus
doubleClickTime	freezeOnConfig	iconAutoPlace
iconClick	iconDecoration	iconImageMaximum
iconImageMinimum	iconPlacement	iconPlacementMargin
interactivePlacement	keyBindings	keyboardFocusPolicy
limitResize	lowerOnIconify	maximumMaximumSize
moveThreshold	passSelectButton	positionIsFrame
positionOnScreen	quitTimeout	resizeCursors
restartSettings	startupKeyFocus	transientDecoration
transientFunctions	useIconBox	wMenuButtonClick
wMenuButtonClick2		

What's Next

The next two chapters cover special custom-environment situations. Chapter 7 concerns special hardware-related configurations. Chapter 8 concerns the use of Starbase graphics and the printing of screen dumps.

7

Customizing Special X Environments

Some applications, perhaps yours is one of them, require customization beyond changing colors, choosing clients for your environment, and modifying window manager resources. This chapter describes the following "special" X Window System customizations:

- Using custom screen configurations.
- Using special input devices.
- Changing mouse key actions
- Going mouseless.
- Customizing keyboard input.
- Creating a custom color database.
- Changing display preferences.
- Compiling `bdf` fonts into `snf` format.
- Using `xrdb` to configure the server.
- Using national languages.

This chapter discusses the following X clients:

xmodmap	Modifies keyboard bindings to modifier keys.
xset	Sets user preferences for display behavior.
rgb	Creates a custom color database.
bdftosnf	Compiles `bdf` format fonts into `snf` format fonts.
xrdb	Initializes the X server with resource specifications.

More information about these clients is in the reference section.

Using Custom Screen Configurations

The default screen configuration for X11 assumes the following about your system:

- You use display 0 (typically the console).
- You have only one physical display screen.
- Your screen uses only one set of pixel planes (the image planes).
- Your screen is at the address node specified by `/dev/crt`.

However, the configuration you actually have may differ from the default configuration. For example, you may be display 1. Or you may be display 0, but have "two heads" (screens). Or you may be screen 0 on display 0, but have two sets of planes, image and overlay. There are many possible combinations of non-default screen configurations.

If you use some configuration other than the default, you must create a custom screen configuration file for yourself.

The Default Screen Configuration File

The default screen configuration file is located in `usr/lib/X11` and is called `X0screens` (x zero screens). The "0" is an arbitrary number chosen to signify the default display.

`X0screens` is an ASCII file. It contains the path and name of the default screen, bitmap device `/dev/crt`.

Creating a Custom 'X*screens' File

To tell the server about a custom screen configuration, you need either to modify `X0screens` or to create an `X*screens` file. The * should be replaced by a display number signifying the new configuration.

Each `X*screens` file represents a custom screen configuration and can contain a maximum of four lines (excluding comments and blank lines).

As an alternative to specifying multiple `X*screens` files, you can make one `X*screens` file for your display. This file will contain the device information for all the configurations. But only one configuration will be in use at a time;

the other configurations will be commented out. To switch configurations, instead of choosing a different `X*screens`, you edit your single configuration file, uncommenting only the configuration you want to use.

Choosing a Screen Mode

Although each `X*screens` file contains lines listing a screen device, the exact syntax of the lines depends on the screen mode you choose. Depending on your system's hardware, you may choose from four screen modes:

image mode	The default screen mode using multiple image planes for a single screen. The number of planes determines the variety of colors available to the screen.
overlay mode	An alternate screen mode using overlay planes for a single screen. Overlay planes provide an alternate (auxiliary) set of planes to the standard image planes. You can see what is in the image planes only if you open a "transparent" window in the overlay plane and move the window over what you want to see. Typically, overlay planes are used in conjunction with image planes in either stacked mode or combined mode.
stacked mode	A combination of image and overlay planes in which a single display has two "logical" screens: image planes and overlay planes. You move between the image and overlay planes by moving the pointer to the edge of the display. Another display pops into view. Typically, the image planes are used for graphics, while the overlay planes are used for text.
combined mode	A combination of image and overlay planes in which a single display has a single screen that is a combination of the image and overlay planes. Starbase or other applications are in windows that can be moved or resized like any other window. Combined mode is available only on the TurboSRX 3-D Display Controller (HP part number 98730A).

Table 7-1.
Graphics Boards, Display Controllers, and Available Screen Modes

With this display hardware ...	You can use these modes ...			
	Image	Overlay	Stacked	Combined
HP 98542A	√			
HP 98543A	√			
HP 98544B	√			
HP 98545A	√			
HP 98547A	√			
HP 98548A	√			
HP 98549A	√			
HP 98550A	√	√	√	
HP 98720A	√	√	√	
HP 98730A	√	√	√	√

Syntax for 'X*screens' File Lines

The syntax for each line of an `X*screens` file is as follows:

```
/dev/device [ { depth { 8 } [doublebuffer] } ]  [# comment]
            [ { depth { 24}                 } ]
            [ { depth 16 doublebuffer       } ]
            [ monitorsize { numberin } ]
                          { numbermm }
```

/dev/*device* Specifies the name of the device file that the X server should read for this display.

depth Specifies the number of image and overlay planes available to the server (one pixel per plane).

doublebuffer	Specifies **double buffer.** Double buffering divides video memory into halves and displays one half while drawing the other. Double buffering is used specifically with graphics programs that double buffer their screen output. This avoids "flashing" during screen redraw.
depth 16 doublebuffer	Specifies the division of the image planes into two 8-bit, double buffered halves.
monitorsize	Specifies the size of the monitor in inches or millimeters.

Determining the Number of Screen Devices

Each line of the text lists a separate screen device (except in combined mode). A screen device can be either a physical device, the CRT screen, or the image planes or overlay planes of a physical device.

For example, if you have a system that includes two physical display screens, you should create an `X*screens` file that contains two lines, one for each physical screen. If you have one physical display screen that is divided into image and overlay planes, you should create an `X*screens` file that also contains two lines. However, the lines will be different.

Mouse Tracking with Multiple Screen Devices

The mouse pointer tracks from left to right. For multiple-plane configuration files, the order of entry determines the tracking order of the mouse pointer. The first line in the file is the device on which the pointer appears when you start X.

Other lines correspond to the screens which appear when the mouse is moved to the right or left side of the current screen.

Thus the order of the lines is important because it tells the server to which screen to move the pointer.

Making a Device Driver File

When you specify a device in a screen configuration file, include the path, usually `/dev`. The device you specify must correspond to a device file in that path. If you don't have the appropriate device file, you must make that device

using the HP-UX `mknod` command. See the "Creating Device Files" section of *HP-UX System Administration Manual* for information on `mknod`.

Examples

The following example shows how you might customize several `X*screens` files.

Suppose you have a high-resolution (1280×1024) TurboSRX screen on which you want to run X Windows and Starbase applications. The image plane of this screen is accessed by the device file `/dev/crt` for the Series 300 (`dev/crt0` for the Series 800). The overlay plane is accessed by the device file `/dev/ocrt` for the Series 300 (`dev/ocrt0` for the Series 800). You would like to switch between four different screen configurations:

- One screen with X11 running in the image planes (image mode). This configuration is described in the `X0screens` file below.
- Two screens with Starbase running in the image planes and X11 running in the overlay planes (overlay mode). You may have only one Starbase application running in this mode, and you can see it only if you open a "transparent" window to look through the overlay planes. This configuration is described in the `X1screens` file below.
- Two screens running (with X11 running in both) on the same display (stacked mode). One screen runs in the image planes, and the other runs in the overlay planes. You move between the two screens by moving the pointer to the edge of the display. The order in which the screens appear is specified by the order in which their designations appear in the `X*screens` file. Starbase is not normally run in this mode. This configuration is described in the `X2screens` file below.
- One screen with two visuals, one with depth 24, and the other with depth 8 (combined mode). Starbase applications are run in windows that can be moved or resized like any other window. You can have several Starbase applications running at once, each in its own window. This configuration is described in the `X3screens` file below. The order in which the screens are described is unimportant.

To accomplish this, you need to have the following four screen configuration files (the entries are slightly different between the Series 300 and Series 800):

`X0screens` Containing the line:

```
/dev/crt     Series 300
/dev/crt0    Series 800
```

`X1screens` Containing the line:

```
/dev/ocrt     Series 300
/dev/ocrt0    Series 800
```

`X2screens` Containing the lines:

```
/dev/crt     Series 300
/dev/ocrt
/dev/crt0    Series 800
/dev/ocrt0
```

`X3screens` Containing the line:

```
/dev/crt /dev/ocrt depth 24 depth 8      Series 300
/dev/crt0 /dev/ocrt0 depth 24 depth 8    Series 800
```

Note that the first file is the default screen configuration provided with X11. The other two files must be created for the particular situation. The `X1screens` file contains two lines, one for each set of planes. Think of the lines as "stacked" for stacked mode. The `X3screens` file contains one line specifying both planes. Think of the lines as "combined" for combined mode.

An alternate means of achieving the same end is to modify the `X0screens` file (or create a custom file) to contain all the possible configurations, with all but the one to be used commented out. The following example shows such a file for the Series 300:

```
### Default Configuration ###
/dev/crt

### Overlay Screens Mode ###
# /dev/ocrt
```

```
### Stacked Screens Mode ###
# /dev/crt
# /dev/ocrt

### Combined Screens Mode ###
# /dev/crt /dev/ocrt depth 24 depth 8
```

Using this method, the name of the file never changes from `X0screens`. Instead, you edit the file, commenting out lines you don't want and uncommenting lines you do want, to switch screen devices.

Either method will work.

Defining Your Display

The DISPLAY environment variable establishes the host, display number, and screen number to which a system sends bitmapped output. For example, on Series 300 and Series 800 systems, the console is typically display 0, screen 0 by default and output is usually sent there.

However, most X11 clients have a `-display` option that enables you to specify a different host, display number, and screen number on which the client should display its output. (By default, clients display on the system on which they are started.)

Specifying a Display with 'x11start'

As mentioned, you can specify a display for your local system or for individual clients.

The display for your local system, the DISPLAY environment variable, is part of your system environment and is used by all clients started and displayed locally.

When you start X11, the `xinit` client sets the DISPLAY variable to the hostname of your system, display number 0, screen 0.

You can modify this behavior by setting and exporting the DISPLAY variable before executing `x11start`. Your `.profile` or `.login` file is a good place to do that.

Finding the DISPLAY Variable

You can view the current setting of your system's DISPLAY environment variable, by typing the following command line:

```
env (Return)
```

This causes a list similar to the following to display. The list contains the current environment settings for your system. Look for the DISPLAY setting.

```
DISPLAY=hpcvfaa:0.0
HOME=/users/alex
LOGNAME=alex
MAIL=/usr/mail/alex
PATH=/usr/bin/X11:.:./bin:../bin:/bin:/usr/local/bin:/usr/bin:/etc:
SHELL=/bin/csh
TERM=98720
TZ=PST8PDT
WINDOWID=7340052
```

In this example, the DISPLAY variable is set to "hpcvfaa:0.0."

Resetting the DISPLAY Variable

To reset the DISPLAY variable, do one of the following:

- If you use `csh`, type the following:

  `setenv DISPLAY` *host:display:screen* (Return)

- If you use `sh` or `ksh`, type the following:

  `DISPLAY=`*host:display.screen* (Return)
  `export DISPLAY` (Return)

For example, before you started X11 using your custom `X1screens` configuration file, you would issue the following command line for a `csh`:

```
setenv DISPLAY host:1.0 (Return)
```

Making 'X*.hosts' Files for Special Configurations

The default screen configuration file `XOscreens` uses the default X11 remote host file `XO.hosts`. The `XO.hosts` file, you will recall, contains a list of all X11 hosts permitted to access your local server.

Each custom `X*screens` file requires that you make a special `X*.hosts` file. The number represented by the * causes the correct screen and host files to be used together.

An `X*.hosts` file is an ASCII text file containing the hostnames of each remote host permitted to access your local server. The syntax is as follows:

host
host
host

For example, if you were on a network and regularly ran clients on hpcvfaa, hpcvfcc, and hpcvfdd, you would want the following lines in your `X*.hosts` file:

```
hpcvfaa
hpcvfcc
hpcvfdd
```

Note that aliases work as well as hostnames, provided they are valid, that is, commonly known across the network.

Using Special Input Devices

The X Window System has an input device file that the X server reads to find out what input devices it should open and attach to the display. The default input device configuration file is `XOdevices`. You can find it in the `/usr/lib/X11` directory.

The Default 'X0devices' File

The default file shipped with the X Window System contains lines of text, but does not specify any input configuration. Rather, it assumes the default input configuration of one keyboard and one pointer.

If this is your configuration, you may not want to change the contents of the file for three reasons:

- Clients can request and receive the services of an input device *regardless* of whether the device is specified in a device configuration file. Thus, you need not change the `X0devices` file, or create a custom file, even though you have a custom input configuration.
- Non-clients (terminal-based programs) such as Starbase *cannot* receive the services of an input device if the device is specified in the device configuration file. Any device in the device configuration file is opened for use by the X server. Thus, changing the `X0devices` file, or creating a custom file, in order to inform the server about a certain input device may interfere with a non-client's ability to access the device.
- Even if you have other *screen* configurations, you can rely on the default input device configuration without having to create an `X*devices` file to match every `X*screens` file. For example, if you had a custom `X8screens` file, you would not necessarily need an `X8devices` file.

A custom `X*devices` file is required only when you want to tell the *X server* about a custom input device configuration.

How the Server Chooses the Default Keyboard and Pointer

Input devices attach to HP computers through an interface known as the Hewlett-Packard Human Interface Link (HP-HIL). Up to seven input devices can be attached to each HP-HIL. However, if the `X*devices` file does not exist, or does not specify otherwise, the X server recognizes only two devices: a pointer and a keyboard (clients, however, may still recognize other input devices).

The X server uses the following order when choosing a pointer:

1. If the `X*devices` file specifies an input device as the pointer, the X server uses that device as the pointer.
2. If `X*devices` makes no specification, or there is no `X*devices` file, the X server takes the last mouse on the HP-HIL (the mouse farthest from the computer) as the pointer.
3. If the X server can open no mouse, it takes the last pointer device (knob box, graphics tablet, trackball, or touchscreen) on the HP-HIL as the pointer.
4. If the X server can open no pointer device, it takes the last keyboard on the HP-HIL as the pointer as well as the keyboard.
5. If no pointer can be opened, the server will not run.

The X server uses a similar order when determining the keyboard:

1. If the `X*devices` file specifies an input device as the keyboard, the X server uses that device as the keyboard.
2. If `X*devices` makes no specification, or there is no `X*devices` file, the X server takes the last keyboard on the HP-HIL (the keyboard farthest from the computer) as the keyboard.
3. If the X server can open no keyboard, it takes the last key device (buttonbox, barcode reader) on the HP-HIL as the keyboard.
4. If no keyboard can be opened, the server will not run.

Creating a Custom 'X*devices' File

At some point, you may want to instruct the server to open a particular device as the keyboard or pointer or have the server open another input device as an extension of the keyboard or pointer. Additional devices with keys are treated as extensions to the keyboard; additional devices that point are treated as extensions to the pointer.

To tell the server about a non-default input device configuration, you must add a device specification line to the appropriate `X*devices`. For example, you would use `X0devices` if you used `X0screens` and `X2devices` if you used `X2screens`.

`X*devices` files are ASCII text files. You can use any ASCII text editor to modify them. Similar to `X*screens` files, you add a device line to the file for each input device you want the server to know about.

Syntax

The device specification lines that you add to the `X*devices` file can have any of three syntaxes.

The Syntax for Device Type and Relative Position. The following syntax uses device type and relative position on the HP-HIL to specify input devices:

relativeposition devicetype use [**#** *comments*]

relativeposition	Specifies the position of a device on the HP-HIL relative to other input devices of the same kind.
devicetype	Specifies the type of input device.
use	Specifies whether the device is the keyboard, the pointer, or has some other use.

Valid positions, types, uses, and examples are in the following section, "Selecting Values for 'X*devices' Files."

Separate the parts of your entry with tabs or spaces. The position of an input device on the HP-HIL is relative to other devices of that type. Thus, `first` means the device connected closest to the computer on the HP-HIL of any device *of that type.*

This syntax is most useful for systems running a single X server with no other programs directly opening input devices. Here, if you add a new input device to the HP-HIL, you don't have to edit the `X*devices` file unless the device is of the same type as one already named in the file and you add the new device ahead of that existing device.

This syntax may become ambiguous, however, when more than one X server is running on the same system or when non-client programs directly access input devices. This is because `first` actually means first device of that type *available* to the server. Thus, a device may be physically first on the HP-HIL, but not

first for the server if the device is unavailable because it is currently being used by some other program.

The Syntax for Device File Name. The following syntax uses the device file name to specify input devices:

/path/devicefile *use* [**#** *comments*]

/path/devicefile — Specifies the path and device file to use as the input device.

use — Specifies whether the device is the keyboard, the pointer, or has some other use.

This syntax is unambiguous when several X servers are running on the same computer or when non-client programs directly access the input device.

The Syntax for Reconfiguring the Path to Device Files. The default path to the device files is `/dev`, but you can specify another path if you choose. Also, if you have more than one HP-HIL, you can specify which HP-HIL the server should use.

The syntax for this is as follows:

path **hil_path** [**#** *comments*]

path — Specifies the path to the device files.

The path specified is handled differently depending on whether your system is an System 300 or an System 800. System 300s only have a single HP-HIL, so no matter what path you specify, the server always checks *only that path* HP-HIL for input devices to open. For example, if you specify `/tmp/foo hil_path` in your `X*devices` file, the server would attempt to open devices `/tmp/foo1` through `/tmp/foo7` on the HP-HIL.

However, System 800s can have up to four HP-HILs numbered 0 through 3. For the System 800s, the HIL number you specify is the *only* place the server checks for input devices to open. Thus if you specify `/dev/hil_2 hil_path`, the server would try to open input devices 1 through 7 on HP-HIL 2 and, if that failed, would not open any input devices at all. If you specified a path that did not end with a digit in the range 0-3, the server would search four

HP-HILs for devices to open using that path as the root of the device file names. For example, if you specified `/tmp/foo hil_path`, the server would attempt to open device files `/tmp/foo0.1` through `/tmp/foo0.7`, `/tmp/foo1.1` through `/tmp/foo1.7`, and so on through `/tmp/foo3.7`

Selecting Values for 'X*devices' Files

`X*devices` files use the following special names for positions, devices, and uses:

Table 7-2. Special Names for 'X*devices' Files

Positions	Device Type	Device Class	HP Part Number	Uses
first	keyboard	keyboard	46021A*	keyboard
second	mouse	pointer	46060A*	pointer
third	tablet	pointer	46087A*	other
fourth	buttonbox	keyboard	46086A*	
fifth	barcode	keyboard	92916A*	
sixth	one_knob	pointer	46083A*	
seventh	nine_knob	pointer	46085A*	
	quadrature	pointer	46094A*	
	touchscreen	pointer	35723A*	
	trackball	pointer	80409A*	
	null			
* or equivalent				

The nine-knob box appears to the X server as three separate input devices. Each row of knobs is a separate device with the first device being the bottom row.

Note also that the HP barcode reader has two modes: keyboard and ASCII. The modes are set via switches on the reader. If you set the barcode reader to ASCII transmission mode, it appears to the server as a barcode reader and the device name is therefore `barcode`. However, if you set the barcode reader to

emulate a keyboard, the barcode reader *appears as a keyboard* and the device name should therefore be `keyboard`. What distinguishes a barcode reader set to keyboard mode from a real keyboard is the relative position or the device filename, depending on which syntax you use.

Similar to the barcode reader, the trackball appears to the server, not as a trackball, but *as a mouse.* Therefore, to specify a trackball, use the `mouse` device name. Again, what specifies the trackball instead of the real mouse is the relative position or the device filename, depending on which syntax you use.

Configuring an Output-Only X Window System

You can create a system on which the X server runs, but which does not have any input devices. In this case, clients could be run from a remote terminal, or from a remote host, and their output directed to the X server.

To create an X11 system with no input, include the following lines in the `X0devices` file:

```
first null   keyboard
first null   pointer
```

Examples

Suppose your input devices include a graphics tablet, a keyboard, and another graphics tablet, and you want to use the tablet closest to the computer as the pointer. You would have the following lines in your `X*devices` file:

```
first    tablet     pointer     The pointer.
second   tablet     other       An extension of the pointer.
first    keyboard   keyboard    The keyboard.
```

In this example, input from the second graphics tablet would appear as an extension of the input from the pointer device, the first tablet.

Now suppose you add another keyboard and a barcode reader (set to ASCII mode) to the above configuration. Also suppose you want the keyboard farthest from the computer to be the keyboard device, with the barcode reader serving as an extension to it. You would have the following lines in your `X*devices` file:

```
first     tablet      pointer     The pointer.
second    tablet      other       An extension of the pointer.
first     keyboard    other       An extension of the keyboard.
second    keyboard    keyboard    The keyboard.
first     barcode     other       An extension of the keyboard.
```

The example now includes input from the first keyboard and from the barcode reader as extensions of the input from the second keyboard. To the X server, the input is indistinguishable.

Note that the barcode reader is in ASCII mode in this example. If the barcode reader were in keyboard mode, the last line of the example should read as follows:

```
third keyboard    other
```

In keyboard mode, the barcode reader is merely the third keyboard on the HP-HIL.

Now suppose you add a nine-knob box to the configuration, but only use the first two rows of knobs. You would have the following lines in the input device configuration file (assuming the barcode reader is set to ASCII mode):

```
first     tablet      pointer     The pointer.
second    tablet      other       An extension of the pointer.
first     keyboard    other       An extension of the keyboard.
second    keyboard    keyboard    The keyboard.
first     barcode     other       An extension of the keyboard.
first     nine_knob   other       Bottom row, pointer extension.
second    nine_knob   other       Middle row, pointer extension.
```

Note that specifying an `other` input device in an `X*devices` file has certain consequences. Each input device you specify as `other` in `X*devices` is opened *exclusively* by the X server. This means that the device is available for clients, but *is not available* for direct access by non-client programs. Since it isn't necessary to list `other` devices in `X*devices` for clients to access them, it may

be better to omit **other** devices from your `X*devices` file. Include them only if no Starbase or other non-client programs access them directly.

Changing Mouse Button Actions

Normally, the mouse pointer buttons are mapped as follows:

Table 7-3. Default Mouse Button Mapping

To press this ...	On a 2-button mouse press this ...	On a 3-button mouse press this ...
Button 1	Left button	Left button
Button 2	Both buttons simultaneously	Middle button
Button 3	Right button	Right button
Button 4		Left and middle buttons simultaneously
Button 5		Middle and right buttons simultaneously

However, it is possible to change these mappings. Possible reasons for changing the mapping are:

- Configuring the mouse for a left-handed person.
- Matching OSF/Motif key mapping or other existing key mappings.

Note

The default mouse button mapping will change to the OSF/Motif mapping in a future release of the X Window System. If you're a System Administrator or application programmer, keep this in mind when designing applications.

Table 7-4. Alternative Mouse Button Mappings

To press	Left Hand Mapping		OSF/Motif Mapping	
Button	**2-button mouse**	**3-button mouse**	**2-button mouse**	**3-button mouse**
Button 1	Right button	Right button	Left button	Left button
Button 2	Both buttons simultaneously	Middle button	Right button	Middle button
Button 3	Left button	Left button	Both buttons simultaneously	Right button
Button 4		Middle and right buttons simultaneously		Left and middle buttons simultaneously
Button 5		Middle and left buttons simultaneously		Right and middle buttons simultaneously

The `xmodmap` utility is used to change mouse button mappings. (`xmodmap` is also used to change keyboard mappings. That topic is discussed later in this chapter.)

Changing Mouse Button Mapping with 'xmodmap'

The syntax for changing mouse button mappings with `xmodmap` is:

$$\texttt{xmodmap} \begin{bmatrix} \texttt{-e "} \begin{bmatrix} \texttt{pointer = default} \\ \texttt{pointer = } \textit{number[number...]} \end{bmatrix} \texttt{"} \\ \texttt{-pp} \end{bmatrix}$$

-e Specifies a remapping expression. Valid expressions are covered in "Customizing Keyboard Input" later in this chapter.

default Set mouse keys back to default bindings

number Specifies a list of button numbers to map the mouse keys to. The order that the numbers appear in refers to the original button mapping.

pp Print the current pointer mapping.

For example, to reverse the positions of buttons 1 and 3 for left-handed mapping:

```
xmodmap -e "pointer = 3 2 1"        2-button mouse
xmodmap -e "pointer = 3 2 1 5 4"    3-button mouse
```

To establish OSF/Motif-standard button mapping:

```
xmodmap -e "pointer = 1 3 2"        2-button mouse
xmodmap -e "pointer = 1 3 2 4 5"    3-button mouse
```

Going Mouseless with the 'X*pointerkeys' File

Your work situation may lack sufficient desk space to adequately use a mouse pointer. You may, therefore, want to "go mouseless" by naming the keyboard (or some other input device) as the pointer.

To go mouseless, you need to have the proper configuration specified in the `X*devices` file and to have a special configuration file named `X*pointerkeys`. The default `X*pointerkeys` file is named `X0pointerkeys`. You can find it in the `/usr/lib/X11` directory. In light of your experience with `X0screens` and `X0devices`, you will probably recognize this as no mere coincidence.

The `X*pointerkeys` file enables you to specify the following:

- The keys that move the pointer.
- The keys that act as pointer buttons.
- The increments for movement of the pointer.
- The key sequence that resets X11.
- The pixel threshold that must be exceeded before the server switches pixel planes in stacked screen mode.

Configuring 'X*devices' for Mouseless Operation

If you have only one keyboard and no pointer devices on the HP-HIL, and you want the keyboard to serve as both keyboard and pointer, you don't have to change the default configuration of `X0devices`. The default input device configuration automatically assigns the pointer to the keyboard if a pointer can't be opened by the server.

If you have two or more input devices, you may need to explicitly specify which device should be the keyboard and which the pointer.

The Default Values for the 'X*pointerkeys' File

By default, when you configure your keyboard as the pointer, the X server chooses certain number pad keys and assigns them mouse operations. Some number pad keys are assigned to pointer movement; other number pad keys are assigned to button operations.

If you don't need to change the pointer keys from their default specifications, you don't need to do anything else to use your keyboard as both keyboard and pointer. However, if you need to change the default pointer keys, you must edit the `X0pointerkeys` file or create a new `X*pointerkeys` file. The `X*pointerkeys` file is the file that specifies which keys are used to move the pointer when you use the keyboard as the pointer.

The default key assignments are listed in the tables in the following section on customizing the `X*pointerkeys` file.

Creating a Custom 'X*pointerkeys' File

You need to modify the existing `X0pointerkeys` file only if the following statements are true:

- You want to use the keyboard for a pointer.
- You want to change the pointerkeys from their default configuration.
- You use the `X0screens` file to configure your display.

You need to create a custom `X*pointerkeys` file only if the following statements are true:

- You want to use the keyboard for a pointer.

- You want to change the pointerkeys from their default configuration.
- You use a configuration file other than the **X0screens** file to configure your display.

Syntax

You assign a keyboard key a mouse function (pointer movement or button operation) by inserting a line in the **X*pointerkeys** file. One line for each action. Lines in the **X*pointerkeys** file have the following syntax:

function *keyname* [# *comment*]

Assigning Mouse Functions to Keyboard Keys

You can assign any mouse function, either a pointer movement or a button operation, to any keyboard key. However, you should first make sure that the key you are thus changing the meaning of doesn't already serve some absolutely vital function.

You can assign keyboard keys to pointer directions by specifying options in an X*pointerkeys file. The following table lists the pointer movement options, the X*pointerkeys functions that control them, and their default values:

Table 7-5. Pointer Movement Functions

To do this ...	Use this function ...	The default key is ...
Move the pointer to the left.	`pointer_left_key`	keypad_1
Move the pointer to the right.	`pointer_right_key`	keypad_3
Move the pointer up.	`pointer_up_key`	keypad_5
Move the pointer down.	`pointer_down_key`	keypad_2
Add a modifier key to the pointer direction keys.	`pointer_key_mod1`	no default
Add a second modifier key to the pointer direction keys.	`pointer_key_mod2`	no default
Add a third modifier key to the pointer direction keys.	`pointer_key_mod3`	no default

Note that the pointer direction keys are the key*pad* number keys on the right side of the keyboard, not the key*board* number keys above the text character keys.

You can assign keyboard keys to pointer distances by specifying options in a `XOpointerkeys` file. The following table lists the options that determine the distance of pointer movements, the `X*pointerkeys` functions that control them, and their default value:

Table 7-6. Pointer Distance Functions

To do this ...	Use this function ...	The default value is ...
Move the pointer a number of pixels.	`pointer_move`	10 pixels
Move the pointer using a modifier key.	`pointer_mod1_amt`	40 pixels
Move the pointer using a modifier key.	`pointer_mod2_amt`	1 pixel
Move the pointer using a modifier key.	`pointer_mod3_amt`	5 pixels
Add a modifier to the distance keys.	`pointer_amt_mod1`	no default
Add a modifier to the distance keys.	`pointer_amt_mod2`	no default
Add a modifier to the distance keys.	`pointer_amt_mod3`	no default

You can assign keyboard keys to mouse button operations by specifying options in a `X*pointerkeys` file. The following table lists the button operations, the `X*pointerkeys` functions that control them, and their default values:

Table 7-7. Button Operation Functions

To do this ...	Use this function ...	The default key is ...
Perform button 1 operations.	`pointer_button1_key`	keypad_*
Perform button 2 operations.	`pointer_button2_key`	keypad_/
Perform button 3 operations.	`pointer_button3_key`	keypad_+
Perform button 4 operations.	`pointer_button4_key`	keypad_-
Perform button 5 operations.	`pointer_button5_key`	keypad_7

You can change the mapping of buttons on the pointer by using options in the `X*pointerkeys` file. The following table lists the `X*pointerkeys` functions that control button mapping and their default values. Like `xmodmap` and `xset`, these functions affect only the X pointer, not any extension input devices.

Table 7-8. Button Mapping Functions

To do this ...	Use this function ...	The default key is ...
Set button 1 value	`button_1_value`	1
Set button 2 value	`button_2_value`	2
Set button 3 value	`button_3_value`	3
Set button 4 value	`button_4_value`	4
Set button 5 value	`button_5_value`	5

You can change the key sequence that exits the X Window System. Also, if you use both image and overlay planes, you can change the distance you must move the pointer before you switch planes. The following table lists these

options, the `X*pointerkeys` functions that control them, and their default values:

Table 7-9. Reset and Threshold Functions

To do this ...	Use this function ...	The default key is ...
Exit the X Window System	`reset`	break
Add a modifier to the exit key.	`reset_mod1`	control
Add a modifier to the exit key.	`reset_mod2`	left_shift
Add a modifier to the exit key.	`reset_mod3`	no default
Set the threshold for changing between image and overlay planes in stacked mode.	`screen_change_amt`	30 pixels

The `screen_change_amt` enables you to avoid switching from one set of pixel planes to another if you accidentally run the pointer off the edge of the screen. The `screen_change_amt` option establishes a "distance threshold" that the pointer must exceed before the server switches pixel planes. As the previous table shows, the default width of the threshold is 30 pixels, but acceptable values range from 0 to 255.

Examples

For example, a common change that you can easily make is to change the keyboard's [▲], [▼], [◀], and [▶] keys to be the pointer direction keys. Press the [▲] key and the pointer moves up. This makes perfect sense. Or at least it does until you try to move the text cursor in your `hpterm` window. If you reassign the arrow keys to the pointer, they will no longer work for the cursor.

Fortunately, the X Window System enables you to pick up to three keys from among the two [Shift] keys, the two [Extend char] keys, and the [CTRL] key and use them each as a **modifier key.** A modifier key is a key that, when you hold it down and press another key, changes the meaning of that other key.

Modifier keys in the **X*pointerkeys** file have three functions:

- They specify that a certain operation can't take place until they are pressed.
- They enable you to adjust the distance covered by the pointer during a movement operation.
- They enable you to change the key sequence that exits you from X11.

For example, you can overcome the problem in the last example by assigning the Left Shift key as a modifier to the pointer direction keys. Now, to move the *hpterm cursor* to the right, you press ▶ as usual. To move the *x server pointer* to the right, you press Left Shift ▶.

Specifying Pointer Keys

The following table lists the valid keynames to use when assigning keyboard keys to mouse functions:

Table 7-10. Valid Pointer Keynames for Hp 46021A Keyboards

Typewriter Keys:					
1	A	K	U	left_shift	return
2	B	L	V	left_extend	,
3	C	M	W	-	.
4	D	N	X	=	/
5	E	O	Y	backspace	right_shift
6	F	P	Z	[	space_bar
7	G	Q	‘	]	right_extend
8	H	R	tab	\	
9	I	S	caps_lock	;	
0	J	T	control	’	
Function Keys:					
f1	f3	f5	f7	blank_f9	blank_f11
f2	f4	f6	f8	blank_f10	blank_f12
Keypad Keys:					
keypad_1	keypad_4	keypad_7	keypad_0	keypad_+	keypad_comma
keypad_2	keypad_5	keypad_8	keypad_*	keypad_-	keypad_tab
keypad_3	keypad_6	keypad_9	keypad_/	keypad_enter	keypad_period
Special Keys:					
enter	stop	clear_line	delete_line	home_cursor	cursor_up
escape	menu	clear_display	insert_char	prev	next
break	system	insert_line	delete_char	select	cursor_left
cursor_down	cursor_right				

Examples

If you only have one keyboard and no mouse, and you can live with the default pointer key assignations, you don't have to do anything else to configure your system for mouseless operation. To move the pointer to the left 10 pixels, you would press the 1 key on the keypad. To press mouse button 1 you would press the * key on the keypad.

However, suppose you wanted to move only one pixel to the left. Although the default value of `pointer_mod2_amt` is one pixel, no key is assigned to the modifier for that amount. Thus, you would need to edit the `XOpointerkeys` file (or create an `X*pointerkeys`) to include a line assigning one of the modifier keys to `pointer_amt_mod2`. The following line in `XOpointerkeys` assigns the Left Shift key to `pointer_amt_mod2`:

```
###pointerfunction       key
pointer_amt_mod2      left_shift
```

Or suppose you wanted to set up your `XOpointerkeys` file so that you could move 1, 10, 25, and 100 pixels. The following lines show one way to specify this:

```
###pointer function       key
pointer_amt_mod1       left_extend
pointer_amt_mod2       left_shift
pointer_amt_mod3       control
pointer_move           1_pixels
pointer_mod1_amt       10_pixels
pointer_mod2_amt       25_pixels
pointer_mod3_amt       100_pixels
```

With these lines in effect, one press of the 1 key on the keypad moves the pointer 1 pixel to the left. Pressing the left Extend char and 1 moves the pointer 10 pixels to the left. Pressing Left Shift 1 moves the pointer 25 pixels to the left. And pressing CTRL 1 moves the pointer 100 pixels to the left.

Or, take the case previously mentioned where you want to use the arrow keys for both text cursor and mouse pointer. You could insert the following lines in your **XOpointerkeys** file:

```
###pointer function      key
pointer_key_mod1       left_shift
pointer_left_key       cursor_left
pointer_right_key      cursor_right
pointer_up_key         cursor_up
pointer_down_key       cursor_down
```

The above lines enable you to use the arrow keys for cursor movement, while using the shifted arrow keys for pointer movement. Note that it is the (Left Shift) key only (not the (Right Shift)) that modifies the press of an arrow key from cursor to pointer movement.

Now take this scenario a step further. Suppose you want to use the arrow keys to operate the pointer, and you also need the arrow keys to control the cursor in an **hpterm** window, but, as luck would have it, another program you frequently operate uses the shift-arrow key sequence to control its cursor.

The easiest way to solve this dilemma is to call in another modifier. The following lines illustrate this. Compare them to the previous example.

```
###pointer function      key
pointer_key_mod1       left_shift
pointer_key_mod2       left_extend
pointer_left_key       cursor_left
pointer_right_key      cursor_right
pointer_up_key         cursor_up
pointer_down_key       cursor_down
```

In this example,

- Pressing the (▲) key moves the **hpterm** *text cursor* up.
- Pressing (Left Shift) (▲) moves the *cursor* up in the program you frequently operate.
- Pressing (Left Shift) (Left Extend char) (▲) moves the *pointer* up.

Using a similar technique, you can also reassign the (CTRL) (Left Shift) (Reset) sequence that exits X11. You can specify the press of a single key to exit X11,

or a combination of two, three, or four key presses. Just make sure that the key sequence you select isn't something you're going to type by accident.

Customizing Keyboard Input

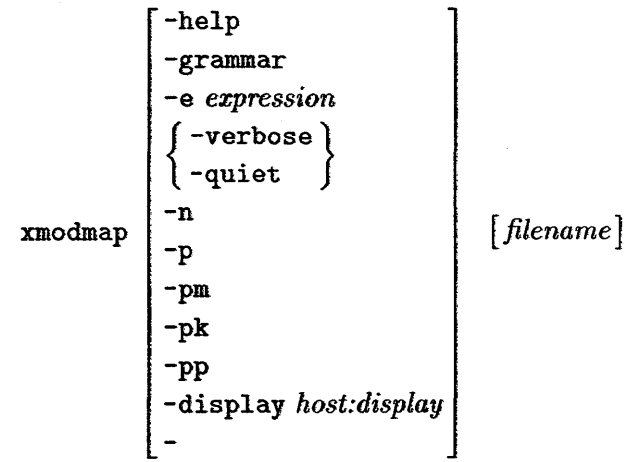
Besides remapping the mouse's pointer and buttons to your keyboard, you can remap any key on the keyboard to any other key.

Modifying Modifier Key Bindings with 'xmodmap'

To change the meaning of a particular key for a particular X11 session, or to initialize the X server with a completely different set of key mappings, use the `xmodmap` client.

Syntax and Options

The syntax for `xmodmap` is as follows:



-display Specifies the host, display number, and screen to use.

-help Displays a brief description of `xmodmap` options.

-grammar	Displays a brief description of the syntax for modification expressions.
-verbose	Prints log information as **xmodmap** executes.
-quiet	Turns off verbose logging. This is the default.
-n	Lists changes to key mappings without actually making those changes.
-e	Specifies a remapping expression to be executed.
-pm, -p	Prints the current modifier map to the standard output. This is the default.
-pk	Prints the current keymap table to the standard output.
-pp	Print the current pointer map to the standard output.
-	Specifies that the standard input should be used for the input file.
filename	Specifies a particular key mapping file to be used.

Specifying Key Remapping Expressions

Whether you remap a single key "on the fly" with a command-line entry or install an entire new keyboard map file, you must use valid expressions in your specification, one expression for each remapping.

A valid expression is any one of the following:

Table 7-11. Valid 'xmodmap' Expressions

To do this ...	Use this expression ...
Assign a key symbol to a keycode.	`keycode` *keycode* = *keysym*
Replace a key symbol expression with another.	`keysym` *keysym* = *keysym*
Clear all keys associated with a modifier key.	`clear` *modifier*
Add a key symbol to a modifier.	`add` *modifier* = *keysym*
Remove a key symbol from a modifier.	`remove` *modifier* = *keysym*

keycode — Refers to the numerical value which uniquely identifies each key on a keyboard. Values may be in decimal, octal, or hexadecimal.

keysym — Refers to the character symbol name associated with a keycode, for example, KP_Add.

modifier — Specifies one of the eight modifier names.

The following are the modifier names available for use in keyboard customization:

Table 7-12. Valid Modifier Names

Modifier Names			
Shift	Control	Mod2	Mod4
Lock	Mod1	Mod3	Mod5

On Hewlett-Packard keyboards, the `mod1` modifier is set to the Extend char keys (Meta_L and Meta_R). However, any of the modifiers can be associated with any valid key symbol. Additionally, you can associate more than one key symbol with a modifier (such as Lock = Shift_R and Shift_L), and you can associate more than one modifier with a key symbol (for example, control = Meta_L and Mod1 = Meta_L).

For example, you can press d to print a lower case "d", Shift d to print a capital "D", Extend char d to print something else, and Shift Extend char d to print still something else.

The `xmodmap` client gives you the power to change the meaning of any key at any time or to install a whole new key map for your keyboard. Like remapping the mouse, a little forethought goes a long way.

Examples

Suppose you had the unfortunate habit of hitting the Caps key at the most inopportune moments. You could remove the Caps lock key from the lock modifier, swap it for the f1 key, then map the f1 key to the lock modifier. Do this is by creating a little swapper file that contains the following lines:

```
!This file swaps the [Caps] key with the [F1] key.

remove Lock = Caps_Lock
keysym Caps_Lock = F1
keysym F1 = Caps_Lock
add Lock = Caps_Lock
```

Note the use of the ! in the file to start a comment line. To put your "swapper" file into effect, enter the following on the command line:

```
xmodmap swapper (Return)
```

If you use such a swapper file, you should probably have an unswapper file. The following file enables you to swap back to the original keyboard mapping without having to exit X11:

```
!This file unswaps the [F1] key with the [Caps] key.

remove Lock = Caps_Lock
keycode 0x54 = F1
keycode 0x37 = Caps_Lock
add Lock = Caps_Lock
```

Note the use of the hexadecimal values to reinitialize the keycodes to the proper key symbols. You put your "unswapper" file into effect by entering the following command line:

```
xmodmap unswapper (Return)
```

On a larger scale, you can change your current keyboard to a Dvorak keyboard by creating a file with the appropriate keyboard mappings. Typically, you would keep this as a special file in your home directory, giving it some name like ".keymap." The easiest way to install your Dvorak keyboard map is by including a line in your `.x11start` file like the following:

```
xmodmap .keymap
```

Printing a Key Map

The -pp option prints a list of the key mappings for the current keyboard.

```
xmodmap -pp
```

The list contains the keycode and up to four 2-part columns. The first column contains unmodified key values, the second column contains shifted key values, the third column contains meta (Extend char) key values, and the fourth column contains shifted meta key values.

Each column is in two parts: hexadecimal key symbol value, and key symbol name.

Creating a Custom Color Database with 'rgb'

Depending on your needs, you may want to make your own custom color database modeled after the `rgb.txt` file. The `rgb.txt` file is the source file the `rgb` client uses to compile the two files the server uses for color database information. Thus, if you listed the files in `/usr/lib/X11` that began with `rgb*`, you'd find not only `rgb.txt`, but also the two files `rgb.dir` and `rgb.pag`.

The file `rgb.txt` is the default color data base for the X Window System. The file is a text file and, in case you haven't looked at it, it contains four columns: red value, green value, blue value, and color name. The following lines are from `rgb.txt`. Note that the red, green, and blue values are given as the decimal equivalents of their hexadecimal values.

Table 7-13. Some Lines from 'rgb.txt'

Red	Green	Blue	Color Name
47	47	100	MidnightBlue
35	35	117	navy blue
35	35	117	NavyBlue
35	35	117	navy
35	35	117	Navy
114	159	255	sky blue
114	159	255	SkyBlue

As the above lines illustrate, several lines are sometimes necessary to account for alternate spellings of the same color.

To make a custom color database, start your text editor and open a new file for your database. Another option is to make a copy of `rgb.txt`, giving it a new name, and add, edit, or delete those values.

Use the following steps to add entries to the database:

1. Specify a decimal value for the red aspect of the color.
2. Press the spacebar or Tab.
3. Specify a decimal value for the green aspect of the color.
4. Press the spacebar or Tab.
5. Specify a decimal value for the blue aspect of the color.
6. Press the spacebar or Tab.
7. Specify a color name for the color.
8. Press Return.
9. Repeat steps 1-8 for the other colors in your custom color database.
10. Save your new database file and exit the editor.

To get the other two files, the ones used by the server, use the `rgb` client. The `rgb` client has the following syntax:

`rgb` *outfile* `<`*infile*

where *infile* is the name of your custom database, the text file you created. The `rgb` client will create *outfile*`.dir` and *outfile*`.pag`. Note that if you choose to modify the existing `rgb.txt` (make a backup copy if you do), you must run it through the `rgb` client before any changes take effect.

To put your new color database into effect, you must add it to your `x11start` command line. For example, if your new database is composed of the files `2brite.txt`, `2brite.dir`, and `2brite.pag`, type the following command line to start your X environment:

```
x11start -- -co 2brite Return
```

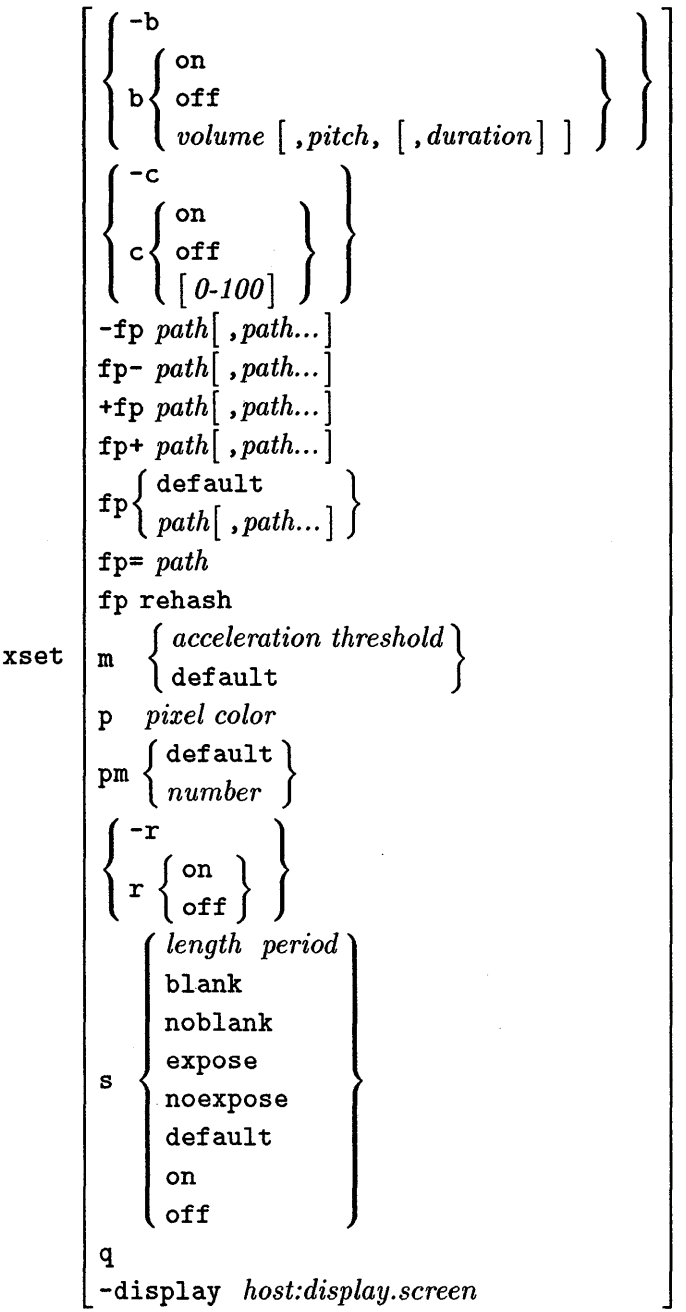
The server by default looks in the `/usr/lib/X11` directory for information and this example assumes that that is where your `2brite*` files are.

Changing Your Preferences with 'xset'

The `xset` client allows you to change your preferences for display options. These preferences last for the length of the X Window System session.

Syntax and Options

The syntax for `xset` is as follows:


xset
-b
b
on
off
volume [,pitch, [,duration]]
-c
c
on
off
[0-100]
-fp path[,path...]
fp- path[,path...]
+fp path[,path...]
fp+ path[,path...]
fp
default
path[,path...]
fp= path
fp rehash
m
acceleration threshold
default
p pixel color
pm
default
number
-r
r
on
off
s
length period
blank
noblank
expose
noexpose
default
on
off
q
-display host:display.screen

-b	Turns the bell off.
b on/off	Turns the bell on or off.
b v[p[d]]	Specifies the bell volume, pitch, and duration. *Volume* is a percentage between 0 and 100 and can be specified without specifying pitch and duration. *Pitch* is in hertz and is specified together with a volume. *Duration* is in milliseconds and is specified with both volume and pitch. If only one parameter is given, it is taken as the volume. If two parameters are given, they are taken as volume and pitch.
-c	Turns the key click off.
c on/off	Turns the key click on or off.
c *0-100*	Specifies the key click volume as a percentage between 0 and 100.
-fp/fp-	Removes the specified directories from the font path.
+fp/fp+	Prefixes or appends the specified directories to the font path (depending on the position of the +).
fp default	Restores the default font path.
fp *path*	Specifies the font path, absolutely.
fp= *path*	Sets the font path.
fp rehash	Causes the server to reread the `fonts.dir` file and the `fonts.alias` files in each path of the server's font path.
m *acceleration threshold*	Specifies the acceleration and threshold of the mouse. *Acceleration* indicates the change in mouse speed (for example: 2=double, 3=triple). *Threshold* indicates the number of pixels of movement required before acceleration takes place. If only one number is given, it is taken as the acceleration.
m default	Resets the mouse acceleration and threshold to their default values.
p *pixel color*	Controls color on a per pixel basis. *Pixel* is an integer representing a specific pixel in the X server's colormap. The

	exact number of pixels in the colormap depends on your hardware. *Color* specifies the color that pixel should be.
pm default	Restores the default font button codes to the pointer map.
pm *number*	Specifies the button codes for pointer map entries.
-r	Turns autorepeat off.
r on/off	Turns autorepeat on or off
s *length period*	Sets the screen saver option on. *Length* is the number of seconds that the server must be inactive before the screen is blanked. *Period* is the number of seconds a particular background pattern will be displayed before changing it.
s blank	Specifies that the screen saver should blank the video, if permitted by your hardware, rather than display the background pattern.
s noblank	Specifies that the screen saver should display the background pattern rather than blank the video.
s expose	Specifies that the server should discard window contents.
s noexpose	Specifies that the server should not enable the screen saver unless it saves window contents.
s default	Sets the system to its default screen saver characteristics.
s on/off	Sets the screen saver feature on or off.
q	Displays the current settings.
-display	Specifies the host, display number, and screen to be reset with `xset`.

Examples

To list the current settings for your system, use the following command line:

```
xset q (Return)
```

The following example speeds up the mouse so that you only have to move it a quarter of the distance you normally would to move the pointer across the screen.

```
xset m 4 2 Return
```

Note

Hardware limitations and implementation differences may affect the results of the `xset` program.

To return to the default settings without leaving X11, type the following command:

```
xset m 1 1 Return
```

Compiling Bitmap Distribution Fonts into Server Natural Format

The X Window System fonts that you select and that the server then uses to display text can be in either of two font formats: server compressed format (scf) or server natural format (snf). Both formats function the same. However, the compressed format takes up less storage space on disk but must be uncompressed by the server for use.

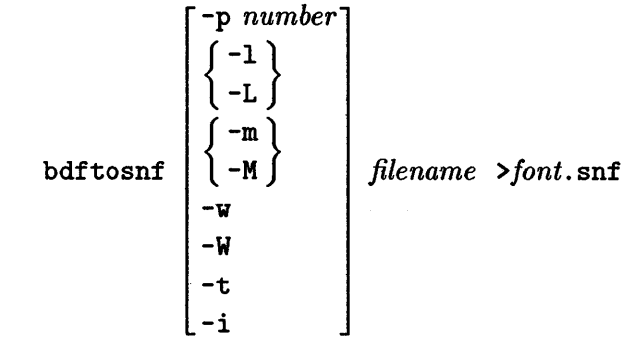
Which format a font is in is signified by the extension that appears after the font file name. A `.scf` signifies a compressed format; a `.snf` signifies a natural (uncompressed) format.

You can compress an snf font file using the HP-UX `compress` command. You can uncompress an scf font file using the HP-UX `uncompress` command.

The X Window System includes a font compiler, `bdftosnf`, which enables you to convert a font in bitmap distribution format (BDF 2.1) into server natural format.

Syntax and Options

The syntax for `bdftosnf` is as follows:



-p	Specifies that font characters should be padded on the right with zeros to the boundary of word *number* where *number* is 1, 2, 4, or 8.
-l	Specifies the output of `bdftosnf` to be least significant byte first.
-L	Specifies the output of `bdftosnf` to be least significant bit first.
-m	Specifies the output of `bdftosnf` to be most significant byte first.
-M	Specifies the output of `bdftosnf` to be most significant bit first.
-w	Print warning if the character bitmaps have bits set to one outside of their defined widths.
-W	Print warning for charcters with an encoding of -1. The default is to ignore such characters.
-t	Expand glyphs in "terminal emulator" fonts to fill the bounding box.
-i	Don't correct ink metrics for "terminal emulator" fonts.
filename	Specifies the name of the bitmap distribution format font to convert to server natural format.

Note that **bdftosnf** by default sends output to standard output (typically the screen). To capture the output as a **.snf** file, therefore, you must redirect the output as shown in the above syntax. Be sure to include the **.snf** filename extension.

Example

The following example takes a bitmap distribution bitmap font file named **tmrm12b.bdf**, converts it to an snf file, then compresses it into an scf file:

```
compress < tmrm12b.snf > tmrm12b.scf Return
```

Note that the **compress** command does not willingly create a **.scf** file, so you have to **mv** or **cp** the compressed **tmrm12b.snf.Z** file to **tmrm12b.scf**.

Using 'xrdb' to Configure the X Server

If you have no **.Xdefaults** file in your home directory, the **x11start** command uses the **xrdb** client to load **/usr/lib/X11sys.Xdefaults** into the server's RESOURCE_MANAGER property. The RESOURCE_MANAGER property contains the list of resources available to the server including the specifications for client colors, keyboard focus policy, and other, similar configurable resources.

You can use **xrdb** to load a custom resource configuration file into the server's RESOURCE_MANAGER property. This enables you to load a resource file that contains conditional statements. You can define one set of resources for use in one particular situation while having another set of resources defined for a different situation.

The benefit of reading the file into the server, instead of having it on disk, is that the server doesn't have to keep reading the file each time you start a new client. Additionally, if you start remote clients and display them on your local screen, the clients will use your local colors. Without a **.Xdefaults** file on the remote host, the clients would appear on your display in their default colors.

How Applications Get their Attributes

The X Window System uses multiple configuration files. An application can get its color and other attributes from several different files. Therefore, how an application gets its attributes (for example, its foreground color) might, on occasion, seem a little mysterious.

Where to Find Attributes

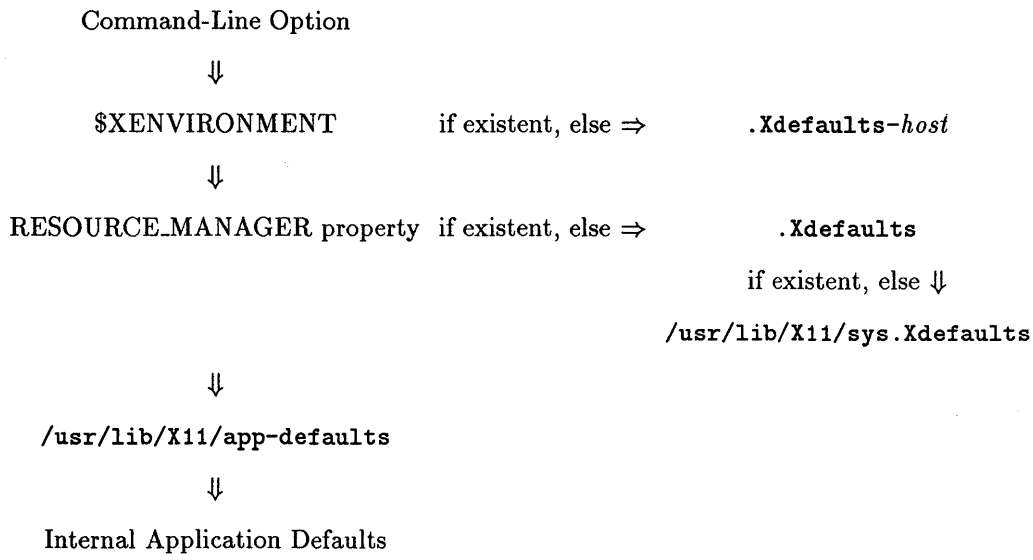
You can find application attributes specified in any of the following places:

- A command line may contain attribute options. These options are good for only that one instance of the application. A command-line option is the equivalent of a *client.resource* statement in a `.Xdefaults` file.
- The `XENVIRONMENT` environment variable, if present, may contain the name of a file that specifies application attributes.
- A `.Xdefaults-`*host* file in your home directory may contain application attributes to be used for a specific remote host. A `.Xdefaults-`*host* file is only read if no `XENVIRONMENT` variable exists.
- The RESOURCE_MANAGER property of the server may contain attributes from a file loaded into the server with `xrdb`.
- A `.Xdefaults` file in your home directory, or a `sys.Xdefaults` file in `/usr/lib/X11` may contain application attributes to be used on your local system. The `.Xdefaults` file is read only if no file has been loaded into the RESOURCE_MANAGER property. The `sys.Xdefaults` file is only read if no `.Xdefaults` file exists in your home directory.
- The `/usr/lib/X11/app-defaults` directory may contain application-specific configuration files that specify attributes for a particular application. An `app-defaults` file is the equivalent of a *Class*resource* statement in an `.Xdefaults` file.
- An application may have internal defaults that specify attributes when no resource configuration files exist.

The following figure presents the hierarchy of resource configuration files. A resource in a source to the top or left of the figure will override the same resource in a source lower or more to the right. For instance, a resource specified in the `.Xdefaults` file will override a resource in the `app-defaults`

file; but a resource specified in the command line will override the `.Xdefaults` specification.

Figure 7-1. The Heirarchy of Resource Configuration Files



Another way to view this figure is to note that the figure is additive from top to bottom, but exclusive from left to right. In other words, if a resource isn't specified on the command line, it is added from the $XENVIRONMENT variable, if set. If set, no resource specifications come from `.Xdefaults-`*host*. Likewise, the resource specification could come from the RESOURCE_MANAGER property; and if so, no resource specifications come from `.Xdefaults`.

Class Struggle and Individual Identity

Additionally, the visible outcome of any attribute specification is influenced by whether that specification is for an individual resource or for a class of resources.

An individual resource begins with a lowercase letter. For example, `foreground` refers to the foreground resource. A class resource, however,

begins with an uppercase letter. For example, `Foreground` refers to the the entire class of foreground resources.

Thus, if no other specifications overruled, the line `*foreground: blue` in your `.Xdefaults` file would make all foregrounds blue. However, the line `*Foreground: blue` would make all resources that belonged to the Foreground class blue. This would include such resources as foreground; cursorColor; pointerColor; bottomShadowColor for softkeys, mwm, icons, and mattes; clock hands; and highlight.

The Order of Precedence Among Attributes

In general, follow this rule of thumb to determine the effect of a resource specification:

A more detailed specification takes precedence over a less detailed specification.

For example, suppose you included the following lines in your `.Xdefaults` file:

```
*Foreground:            red
HPterm*Foreground:      DarkSlateGray
HPterm*foreground:      coral
HPterm*cursorColor:     green
```

The first line makes all resources of the class Foreground red. The second line overrules the first line, but *only* in the case of clients of class HPterm (of which there is only one—the `hpterm` client itself). Line two makes the Foreground class resources of all `hpterm` clients DarkSlateGray. Lines three and four give `hpterm` clients coral foregrounds and green cursors respectively, while the other resources of class Foreground (pointerColor, cursorColor, softkey foreground and bottomShadowColor, and scrollbar foreground and bottomShadowColor) remain DarkSlateGray for `hpterm` clients.

Naming a Client

Additionally, you can give a client of some class a name. This allows you to allocate resources to that client by class, by client, *and* by name.

For example, `HPterm` is a client class. The `hpterm` window is a client of that class. But, using the following syntax, you can also give an `hpterm` client of the `HPterm` class a *name:*

client.name: *name*

Thus, you could add three lines to the preceding lines and have the following specifications in your .Xdefaults file:

```
*Foreground:            red
HPterm*Foreground:      DarkSlateGray
HPterm*foreground:      coral
HPterm*cursorColor:     green
hpterm.name:            BWterm    #Monochrome hpterm window
BWterm*Foreground:      black
BWterm*Background:      white
```

This illustrates the ability to create a named window, in this case a black and white hpterm window, that overrides the specifications for class and client resources. In this example, all hpterm windows will be named BWterm. To start an hpterm window with another name (for instance special), type hperm -n special.

Syntax and Options

The syntax for xrdb is as follows:

```
                ┌ -help                                          ┐
                │ { -cpp   } path/filename                       │
                │ { -nocpp }                                     │
                │ -symbols                                       │
                │ -query                                         │
                │ { -load path/filename   }                      │
xrdb            │ { -merge path/filename  }                      │  [filename]
                │ { -remove               }                      │
                │ { -edit path/filename   }                      │
                │ -backup string                                 │
                │ -Dname[=value]                                 │
                │ -Uname                                         │
                │ -Ipath/directory                               │
                └ -display host:display                          ┘
```

-help	Displays a list of options for **xrdb**.
-display	Specifies the host and display of the server to be loaded with the configuration information.
-query	Displays the current contents of the server's RESOURCE_MANAGER property.
-load	Specifies that **xrdb** should load the file named on the command line into the RESOURCE_MANAGER property, overwriting the current resources listed there. This is the default action.
-merge	Specifies that **xrdb** should load the file named on the command line into the RESOURCE_MANAGER property, merging the new resources with the current resources instead of overwriting them.
-remove	Removes the current configuration file from the RESOURCE_MANAGER property.
-edit	Places the contents of the RESOURCE_MANAGER property into the named file, overwriting resources specified there.
-backup	Specifies a suffix to be appended to the filename used in the **-edit** option to create a backup file.
-cpp	Specifies the path and filename of the C preprocessor to use when loading a configuration file containing **#ifdef** or **#include** statements. **xrdb** works with CPP and other preprocessors as long as they accept the **-D**, **-U**, and **-I** options.
-nocpp	Specifies that **xrdb** should not use a preprocessor before loading the configuration file (the file contains no statements that need preprocessiong).
-symbols	Displays the symbols currently defined for the preprocessor.
-D*name*	Defines a symbol for use with conditional statements in the configuration file used by the RESOURCE_MANAGER property.
-U*name*	Removes a defined symbol from the RESOURCE_MANAGER property.

-I*path/directory* Specifies the search path and directory of `#include` files used in the RESOURCE_MANAGER.

Examples

The `xrdb` client enables you to swap resource configuration files in and out of the X server's RESOURCE_MANAGER property. For example, suppose you need to keep switching between your `.Xdefaults` configuration and a special `/projects/proto.defaults` configuration that contains different color resource specifications. To change to `proto.defaults`, type the following:

```
xrdb -nocpp -load /projects/proto.defaults Return
```

To see that the resources have actually been swapped, type the following:

```
xrdb -query Return
```

Any new clients started now will have the colors specified in `proto.defaults`. To change your existing colors to the `proto.defaults` colors, restart the window manager.

As another example, suppose you are the system administrator of an System 300 diskless cluster and that the cluster includes several different types of monitors. One option that `xrdb` gives you is the ability to create a `/usr/lib/X11/syscus.Xdefaults` file containing resource modules headed by `ifdef` statements—one `ifdef` statement for each of the different monitor types. In each user's `.xllstart` script, you replace the existing line calling `xrdb` with the following:

```
xrdb /usr/lib/X11/syscus.Xdefaults
```

The C language preprocessor reads the class of the monitor (StaticGray, PseudoColor, etc.) passed to it by `xrdb` and, based on that information, select the correct `ifdef` module.

As an elaboration of this, you could include a special module in `syscus.Xdefaults` for novice users (giving them a simplified environment) by placing the following line in their `.x11start` file:

```
xrdb -DUSER=beginner /usr/lib/X11/syscus.Xdefaults
```

This line would select the correct monitor module for the novice's monitor, while tempering the usual resources for that monitor type with resources from the beginner module.

Using Native Language Input/Output

Though most character sets are composed of 8-bit characters, some languages (Japanese, Chinese, and Korean) have larger character sets that require 16-bit characters. The X Window System supports the use of 16-bit character input with the Native Language Input/Output (NL I/O) subsystem.

To use NL I/O you must have the following:

- The NL I/O subsystem properly installed on your system.
- The appropriate language keyboard *or* an ASCII keyboard. A client that uses XHPSetKeyboardMapping allows NL I/O to be used with any language HP keyboard.
- The appropriate NL I/O fonts installed in the `/usr/lib/X11/fonts` correct directory:.

```
/usr/lib/X11/fonts/hp_japanese/
/usr/lib/X11/fonts/hp_chinese_s/
/usr/lib/X11/fonts/hp_chinese_t/
/usr/lib/X11/fonts/hp_korean/
/usr/lib/X11/fonts/hp_katakana/
```

Configuring 'hpterm' Windows for NL I/O

You can configure an `hpterm` window to display NL I/O. The process uses the `config keys` and `terminal config` softkeys available with `hpterm` windows to configure the window for NL I/O. Follow the steps outlined in the *Native Language I/O System Administrator's Guide* (HP 92559-90002) and the *Native Language I/O Access User's Guide* (HP 92559-90001).

This configures the `hpterm` client. You must also select the appropriate NL I/O font.

Specifying an NL I/O Font

NL I/O fonts are part of the NL I/O product. They are installed in the subdirectories of `/usr/lib/X11/fonts` directory when you install your NL I/O subsystem.

You specify an NL I/O font exactly like you specify any other font. For example, if you want to create an `hpterm` window that uses the Japanese font `jpn.8x18`, use the following command line:

```
hpterm -fn jpn.8x18 & Return
```

Where to Go Next

The next chapter discusses X Window System printing and screen dumping utilities. The chapter after that, chapter 9, discusses the X Window System as an environment for Starbase applications.

8

Printing and Screen Dumps

The X Window System includes clients that enable you to do **screen dumps.** A screen dump is an operation that captures an image from your screen and saves it in a bitmap file. You can then redisplay, edit, or send the file to the printer for hardcopy reproduction.

Read this chapter if you need to "take a picture" of something on the screen for future use or if you want to print what is on your screen.

This chapter discusses the following topics:

- Making a screen dump.
- Displaying a screen dump.
- Printing a screen dump.

Making and Displaying Screen Dumps

X11 windows can be dumped into files by using the `xwd` client. The files can be redisplayed on the screen by using the `xwud` client.

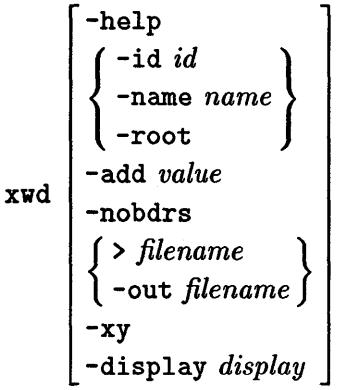
Making a Screen Dump with 'xwd'

The `xwd` client allows you to take a "picture" of a window that is displayed on the screen and store it in a file. The filed picture can then be printed, edited, or redisplayed. You select the window to be dumped either by clicking the mouse on it or by specifying the window name or id on the command line.

The resulting file is called an `xwd`-format bitmap file or an `xwd` screen dump. All of the figures used in this manual are `xwd` screen dumps.

Syntax and Options

The syntax for **xwd** is as follows:



-help	Provides a brief description of usage and syntax.
-id	Specifies the window to be dumped by its *id* rather than using the mouse to select it.
-add	Adds value to every pixel.
-name	Specifies the window to be dumped by its *name* rather than using the mouse to select it.
-root	Specifes that the window to be dumped is the root window.
-nobdrs	Dumps the window without borders.
-out	Specifies that the screen dump is to be stored in the file *filename.*
>	Specified that the screen dump is to be stored in the file *filename.*
-xy	Selects 'XY' format of storage instead of the default 'Z' format.
-display	Specifies the screen that contains the window to be dumped.

Example 1: Selecting a Window with the Pointer

This example stores a window in a file named **savewindow**, using the pointer to determine which window you want.

1. Display the an hpterm or xterm window.
2. Type:

   `xwd -out savewindow` (Return)

   The pointer changes shape, signifying you can select a window to dump.
3. Move the pointer into the window you want to dump. Press and release any pointer button. After the image is captured, the cursor changes back to its normal shape and the window is stored in the file `savewindow`.

Example 2: Selecting a Window with a Name

If you know the name of the window you want to dump, you don't need to use the pointer at all. This example dumps the window named "calendar" to a file named `calendar.dump`.

```
xwd -name calendar -out calendar.dump (Return)
```

Displaying a Stored Screen Dump with 'xwud'

The `xwud` client allows you to display an `xwd`-format file on your monitor. You could have created the file earlier with `xwd` or translated it from another format into `xwd` format.

Note

The image to be restored has to match the depth of the system on which it is to be restored. For example, an image created and stored using a depth of four cannot be restored on a system with a different depth.

Syntax and Options

The syntax for `xwud` is as follows:

```
xwud [-help
      -in filename
      -inverse
      -display host:display.screen]
```

-help Displays a brief description of the options.

-in	Specifies the file containing the screen dump.
-inverse	Reverses black and white from the original monochrome dump.
-display	Specifies the screen on which to display the dump.

Example

This example displays the bitmap file `myfile`.

```
xwud -in myfile (Return)
```

Printing Screen Dumps

Before you can print the screen dump, you need to ensure that your printer is connected and talking to your computer.

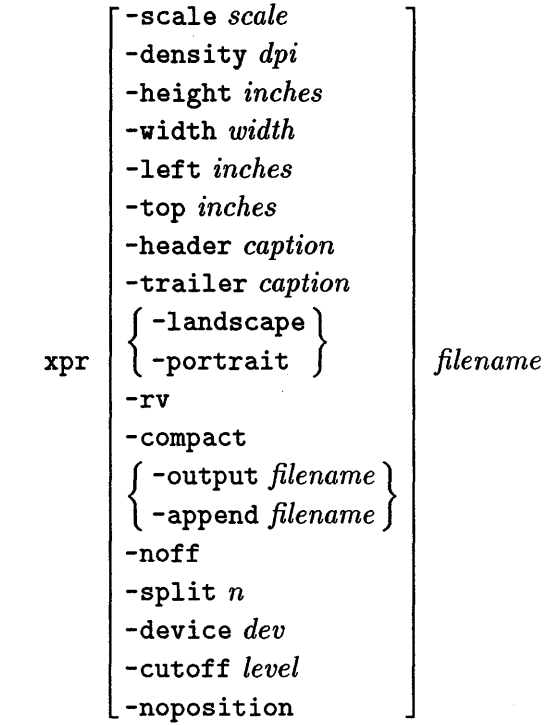
If you are the system administrator, refer to the *HP-UX System Administrator Manual* for information about these tasks. If you're not the system administrator, ask the person who is to perform these tasks:

- Connect the printer to your computer.
- Create a device file for the printer on your computer.
- Run the print spooler.

Printing Screen Dumps with 'xpr'

`xpr` prints a screen dump that has been produced by `xwd`.

Syntax and Options



-scale	Specifies a multiplier for pixel expansion. The default is the largest that will allow the entire image to fit on the page.
-density	Specifies the dots per inch for the printer.
-height	Specifies the maximum height in inches of the window on the page.
-width	Specifies the maximum width in inches of the window on the page.
-left	Specifies the left margin in inches. The default is centered.
-top	Specifies the top margin in inches. The default is centered.
-header	Specifies a caption to print above the window.
-trailer	Specifies a caption to print below the window.

-landscape	Prints the window in landscape mode. The default prints the long side of the window on the long side of the paper.
-portrait	Prints the window in portrait mode. The default prints the long side of the window on the long side of the paper.
-rv	Reverses black and white from the original screen.
-compact	Provides efficient printer directions for a window with lots of white space (PostScript printers only).
-output	Specifies a file to store the output in.
-append	Adds the window to the end of an existing file.
-noff	Specifies that the window should appear on the same page as the previous window. Used with -append.
-split	Prints the window on *n* pages. Not applicable to HP printers.
-device	Specifies the printer to use.

ljet	HP LaserJet series, HP ThinkJet, QuietJet, RuggedWriter, HP2560 series, HP2930 series, other PCL devices.
pjet	HP PaintJet (color mode).
ljetxl	HP PaintJet XL.
ln03	DEC LN03.
la100	DEC LA100.
ps	PostScript printers.
pp	IBM PP3812.

-cutoff	Specifies intensity for converting color to monochrome for printing on a LaserJet printer.
-noposition	Bypasses header positioning, trailer positioning, and image positioning commands for the LaserJet and PaintJet printers.
filename	Specifies the `xwd` file to print.

Example

Suppose you want to print a **xwd** file named **myfile** that you previously created with **xwd**. You want to print the file on a LaserJet printer in portrait mode with black and white the reverse of the original **xwd** file.

```
xpr -device ljet -portrait -rv myfile | lp -oraw (Return)
```

Reversing colors is often used when preparing illustrations for documents. The original illustration can be done in white with a black background, which is easy to see on computer displays, but reversed to give a black drawing on a white background, which is common in printed material.

Moving and Resizing the Image on the Paper

You may not always want to have the image print exactly in the same size or location as the default choices place it.

Sizing Options

The three sizing options for **xpr** are:

-scale	Each bit of the image is translated into a grid of the size you specify. For example, if you specify a scale of 5, each bit in the image is translated into a 5 by 5 grid. This is an easy way to increase the size without refiguring the height and width.
-height	The maximum height in inches of the image on the page.
-width	The maximum width in inches of the image on the page.

The actual printed size could be smaller than -height and -width if other options, such as the orientation ones, conflict with them.

Location Options

The two location options for **xpr** are:

-left	The left margin in inches.
-top	The top margin in inches.

If -left is not specified, the image is centered left-to-right. If -top is not specified, the image is centered top-to-bottom.

Orientation Options

The two orientation options to **xpr** are:

-landscape — The image is printed so that the top of the image is on the long side of the paper.

-portrait — The image is printed so that the top of the image is on the short side of the paper.

If neither option is specified, **xpr** will position the image so that the long side of the image is on the long side of the paper. However, you can force it to print either in landscape mode or portrait mode by using the appropriate option.

Unless told otherwise by the sizing options, **xpr** makes the image as big as necessary to fit in the orientation specified.

Printing Multiple Images on One Page

xpr normally prints each image on a separate page. The -noff option is used to print more than one image on a page.

Printing Color Images

Printing Color Images on a PaintJet Printer

Use the device name `pjet` to direct output to a PaintJet printer.

For example, the following command prints a **xwd** file named `myfile` on a PaintJet.

```
xpr -device pjet myfile Return
```

Printing Color Images on a LaserJet Printer

Color images printed on a LaserJet printer will be in black and white instead of color.

xpr prints only in black and white, no shades of gray. If your original color image contained many colors of the same intensity, the LaserJet version may be all light or all dark. If that happens, use the -cutoff option to change the mapping of color intensities. Anything above the cutoff value is white and anything below is black. Note that the default cutoff value is 50 percent.

If you want color images to print in shades of gray on your Laserjet, use the StarBase utility `pcltrans` instead of `xpr`. Refer to the StarBase documentation for information.

Where To Go Next

Only one chapter left! Chapter 9 covers using the powerful Starbase graphics library from X11.

9

Using Starbase on X11

Starbase is a powerful graphics library from Hewlett-Packard. It provides two-dimensional and three-dimensional graphics, a variety of input and output capabilities, and high performance features, such as hidden surface removal, shading, and light sourcing.

This chapter describes how X11 interacts with Starbase. It does not describe Starbase itself. For detailed information about Starbase features, refer to *Starbase Programming with X11* in the Starbase documentation.

This chapter covers the following topics:

- Using the `X*screens` file to control display options.
- Starting the X11 server.
- Opening and destroying windows for Starbase applications.
- Creating transparent windows.
- Conversion utilities.

Using the X*screens File

This section reviews some concepts that you need to understand before starting Starbase applications:

- `X*screens` file.
- Monitor Type.
- Operating modes.
- Double buffering.

The `X*screens` file is a system file that contains the screen configurations you want to use. Before you run a Starbase application, you should ensure that the configuration is correct for Starbase. The `X*screens` file is described in chapter 7.

The following sections describe options you should be aware of when running Starbase on X11. It also explains how to specify those options in the `X*screens` file. If the `X*screens` file you are using doesn't have the correct entries to do what you want, edit it to include the correct information.

Generally, once you have modified the `X*screens` file, you won't have to change it again unless you add a new monitor.

Monitor Type

Starbase can run on a wide variety of graphics monitors. More sophisticated monitors provide a wider choice of options. The following table shows which options are available for different monitors.

Table 9-1. Display Hardware and Available Options

With this display hardware ...	You can use these options ...			
HP Part Number	**Maximum Planes (colors)**	**Modes**	**Double buffer**	**Depth**
HP 98542A	1 Image (monochrome)	Image		
HP 98543A	4 Image (monochrome)	Image	√	
HP 98544B	1 Image (monochrome)	Image		
HP 98545A	4 Image (monochrome)	Image		
HP 98547A	6 Image (64)	Image	√	
HP 98548A	1 Image (monochrome)	Image		
HP 98549A	6 Image (64)	Image	√	
HP 98550A	2 Overlay (4) 8 Image (256)	Image Overlay Stacked	√	
HP 98720A	3 Overlay (8) 8-24 Image (256 from 16 million)	Image Overlay Stacked	√ √	√ √
HP 98730A	4 Overlay 8-24 Image (256 from 16 million)	Image Overlay Stacked Combined	√ √ √	√ √ √

Operating Modes

Image and Overlay Planes

Monitors can have two kinds of display planes, image and overlay. The image plane allows the monitor hardware to help the graphics commands run faster and more efficiently.

Server Operating Modes

The operating mode results from the way you specify the image and overlay screens in the `X*screens` file.

The four different modes are:

Overlay mode	The X11 server operates only in the overlay planes. Starbase can display in its "raw" mode, writing directly to the image planes, rather than to a window. A "transparent" overlay window can look through to the Starbase display in the image planes. The Starbase double buffering feature does not apply in this mode.
Image mode	This is the only mode available on those displays that do not have overlay planes. Even if overlay planes are available, you may want to use image mode to have a greater number of colors available.
Stacked screen mode	In this mode, the image planes are treated as one screen and the overlay planes as another, separate screen, providing twice as much screen space. The pointer is moved to the edge of the display to switch between the overlay and image planes.
Combined mode	This mode (available only on the HP 98730A) treats the overlay and image planes as a single device that provides multiple window types to client programs.

These modes are also described in chapter 7.

Monochrome monitors and low-level color monitors run in the image mode.

Documentation for the Starbase application program will tell you which mode or which plane the application expects.

The following examples show how `X*screens` entries vary for each mode.

Example 1: Image Mode

This example shows an image mode entry in the `X*screens` file. The same entry is used regardless of the type of monitor. The number of colors available to you depends on the monitor. The entry may also have the options discussed in the next few sections.

```
/dev/crt      Series 300
/dev/crt0     Series 800
```

Example 2: Overlay Mode

This example shows an overlay mode entry in the `X*screens` file for monitors that support overlay mode. The number of colors you are able to use depends on your display adaptor. Only one Starbase application can be run at a time in this mode, and you can see it only if you open a "transparent" window in the overlay plane to look through the overlay plane to the image plane.

```
/dev/ocrt     Series 300
/dev/ocrt0    Series 800
```

Example 3: Stacked Mode

This example shows the entries for stacked mode for monitors that are able to support stacked mode. Starbase applications are not generally run in this mode. Stacked mode is indicated by having each entry on a separate line. Image plane entries can have the options discussed in the next few sections. Note that the order of the entries determines the order of the screens. Screen 0 is the first entry, screen 1 is the second entry, and so on.

```
/dev/crt      Series 300
/dev/ocrt
/dev/crt0     Series 800
/dev/ocrt0
```

Example 4: Combined Mode

This example shows how a combined mode entry is made in the `X*screens` file for monitors that support combined mode. You can have several Starbase applications running at the same time, each in its own window that can be moved or resized like other windows. Combined mode is indicated by having the entries for the overlay and image planes on the same line (it is unimportant which entry is first). Image plane entries can have the options discussed in the next few sections.

```
/dev/ocrt /dev/crt      Series 300
/dev/ocrt0 /dev/crt0    Series 800
```

Double Buffering

This feature does not apply to monochrome monitors or when the X11 server is running in the overlay planes.

Double buffering means that Starbase uses half of the color planes of your monitor to display to the screen, and uses the other half to compute and draw the next screen display. This provides smooth motion for animation, and it is also faster. However, double buffering reduces the number of colors available for displaying on the screen at one time. Some applications require double buffering. If you run a double-buffered application in single buffered mode, the display will flash or flicker rapidly.

The following examples are for the Series 300.

Example 1: Image Mode

```
/dev/crt doublebuffer
```

Example 2: Stacked Mode

```
/dev/ocrt
/dev/crt doublebuffer
```

Example 3: Combined Mode

```
/dev/ocrt /dev/crt doublebuffer
```

Screen Depth

You can specify a screen depth for image planes in the `X*screens` file. Valid depths for regular (single buffer) mode are 8 and 24. Valid depths for doublebuffer mode are 8, 16, and 24. The depth of overlay planes is determined by the `/dev` entry in `X*screens`. The depth for the HP 98550A is 2 overlay planes: the depth for the HP 98720A has 3 overlay planes; and the depth for the HP 98730 can be either 3 or 4 overlay planes.

More planes means more colors can be displayed simultaneously. For computer-generated graphics to look as realistic as photographs, thousands of colors must be shown at the same time. 8 planes means that 2^8 (256) colors can be shown, while 24 planes means that 2^{24} (16 million) colors can be shown.

Note that depth is specified only when you have more than one depth available. This feature is available only on the HP 98720A and HP 98730A Display Controller.

The following examples are for the Series 300.

Example 1: Image Mode

The following example shows an `X*screens` file entry for an HP 98720A monitor running in image mode. Windows can have 8 planes (256 colors) displayed simultaneously.

```
/dev/crt depth 8
```

Example 2: Combined Mode

The following example provides two doublebuffered depths in the image planes: depth 8 (16 planes/2) and depth 12 (24 planes/2). That is, some windows in the image planes could have a depth of 8 planes, while others could have a depth of 12 planes. This is possible only in combined mode.

```
/dev/ocrt /dev/crt depth 16 depth 24 doublebuffer
```

Starting the X11 Server

Once you have ensured that the options you need are in the `X*screens` file, type

`x11start -- :`*n* Return

where *n* is replaced by the number of the `X*screens` file you want to have in effect. For example, if you have all your display options in the `X0screens` file, type `x11start -- :0` to start the server.

Window-Smart and Window-Naive Programs

Window-smart applications are able to create and destroy the windows in which they operate.

Window-naive (sometimes called window-dumb) applications aren't able to create and destroy windows on their own. They need help from the X11 system.

Although this chapter discusses window-smart and window-naive applications in relation to Starbase, the same procedures are used to start non-Starbase programs.

Is My Application Window-Smart or Window-Naive?

If you are using an existing application, the documentation that comes with the application will tell you how to start it. You don't have to worry whether it is window-smart or window-naive, just follow the directions.

If you are writing a new application using Starbase, use the `xwcreate` and `xwdestroy` commands. Rather than typing the commands each time you want to test the new program, put the commands in a file, then execute the file to start the application. In this case, the application is window-naive but the file is window-smart.

Running Window-Smart Programs

From an `hpterm` window, type the name of the Starbase program you want to run.

For example, the following command will start a hypothetical Starbase application named `planetarium` that displays a moving display of the night sky. Assume that the program is in the `/users/funstuff/` directory on your computer.

```
/users/funstuff/planetarium Return
```

Running Window-Naive Programs

Window-naive programs cannot open and close the window they need to run in, so you must do it for them with clients (a terminal emulator, for example). Programs that use the Starbase graphics library are window-naive.

Most window-naive programs are able to run in the X Window System environment using the `sox11` device driver. The `sox11` driver is described in the *Starbase Device Drivers* manual. But window-naive clients still need help to create and destroy the windows they display their output in.

To enable such window-naive graphics programs to run within X, you need four special helper clients to create and destroy the windows used by the naive graphics programs. The clients are:

- `gwind`
- `xwcreate`
- `xwdestroy`
- `gwindstop`

`gwind` runs in the background and services requests from the other three helper clients. When requested by `xwcreate`, `gwind` creates a window in which an application can display its output; when requested by `xwdestroy`, `gwind` destroys the window. *You don't need to start the* `gwind` *program,* `xwcreate` *and* `xwdestroy` *start and stop it for you.*

The next sections cover:

- Creating a window
- Destroying a window

An example showing all of these steps follows the discussion.

Creating a Window with 'xwcreate'

`xwcreate` requests `gwind` to create a window for a window-naive graphics program to use for its output. The graphics program must exist on the same computer that is running `xwcreate`. If `gwind` is not already running when `xwcreate` is executed, `xwcreate` will start `gwind`. Once `xwcreate` has created a window, you can use the window to run your graphics program. When you

finish that application, you can use the same window to run another graphics program if you wish.

When to Use 'xwcreate'

Use `xwcreate` from the command line.

Syntax and Options

`xwcreate` [`-display` *host:display.screen* | `-parent` *parent* | `-geometry` *width*×*height*±*col*±*row* | `-r` | `-bg` *color* | `-bw` *pixels* | `-bd` *color* | `-depth` *depth* | `-wmdr` *directory* | `-title` *name*]

-display	Specifies the screen the window will appear on
-parent	Names a window to be the parent of the window being created.
-geometry	Specifies desired size and location of window.
-r	Specifies backing store. Default is no backing store.
-bg	Specifies the background color. The default is black.
-bw	Specifies the border width in pixels. The default is 3 pixels wide.
-bd	Specifies the border color. The default is white.
-depth	Specifies the depth of the window. The default is the same depth as its parent.
-wmdir	Specifies the name of the directory containing the pty file for the window.
-title	Specifies the name the window will be called.

The `depth` option is where you tell the window manager what set of planes you want the window to be in. If you specify nothing, the window is created with the same depth as its parent, or with the same depth as the root if no parent is specified. If you specify a depth, the window will be placed in the image plane with the depth (number of color planes) you specify.

The following example creates a window named "foo:"

```
xwcreate -title foo Return
```

Destroying a Window with 'xwdestroy'

`xwdestroy` destroys the window created by `xwcreate`. If that window is the only graphics window present at that time, `gwind` will also terminate.

When to Use 'xwdestroy'

Use `xwdestroy` from the command line.

Syntax and Options

`xwdestroy` [`-wmdir` *path/directory*] *window1 window2 ...*

-wmdir — Specifies the directory containing the pty file for the window.

window — Specifies the window or windows to be destroyed.

The following example will destroy a window named "foo:"

```
xwdestroy foo
```

Destroying a Window with 'gwindstop'

`gwindstop` destroys all windows created by `gwind` in the specified directory. If, however, you use `xwdestroy` to remove the *last* window opened for graphics use, `xwdestroy` will terminate `gwind`. You *do not* need to use `gwindstop`.

Caution

You must use `xwdestroy` or `gwindstop` to get rid of a window after you have finished running your graphics application. Do *not* use `kill` to remove the `gwind` process associated with the window. If you should accidentally do so, you must type the command `rm $WMDIR/wm`. Failure to do this will result in `xwcreate` not running the next time you call it.

When to Use 'gwindstop'

Use `gwindstop` from the command line.

Syntax and Options

`gwindstop` [*directory*] [*directory*] ...

directory The directory containing the pty files for the windows to be destroyed.

Running Starbase in Raw Mode

If your monitor doesn't support overlay planes, you can run Starbase in "raw" mode, which means that Starbase writes to the entire screen rather than to a window. You then use a transparent window to see through to the Starbase output.

For information about Starbase raw mode, refer to the Starbase documentation.

Using Transparent Windows

Transparent windows allow you to look through an overlay window into the image planes.

Creating a Transparent Window with 'xseethru'

`xseethru` is a transparent overlay-plane window used to see through the overlay planes to the image planes.

When to Use 'xseethru'

Use `xseethru` from the command line.

Syntax and Options

`xseethru` [`-geometry` *width*×*height*±*col*±*row*
`-display` *host:display.screen*]

-geometry — The geometry used to create the window. Refer to chapter 4 for more information about geometry.

-display — The screen the window will appear on. Refer to chapter 4 for more information about display.

Example

This example opens a transparent window 100-pixels by 100-pixels in size and located 50 pixels from the left and 25 pixels from the top of the screen.

```
xseethru -geometry 100x100+50+25 (Return)
```

Creating a Transparent Window with 'xsetroot'

`xsetroot` allows you to make the root window transparent when you are running X in the overlay planes.

When to Use 'xsetroot'

Use `xsetroot` from the command line.

Syntax and Options

`xsetroot [-solid` *color*`]`

-solid Sets the window color to *color.*

Example

This example turns the root window into a transparent window.

```
xsetroot -solid transparent (Return)
```

Creating a Transparent Background Color

Any window may have `transparent` as its background color.

Chapter 4 explains how to set colors in a window. This example opens an `hpterm` window with a transparent background color.

```
hpterm -bg transparent (Return)
```

Conversion Utilities

This section shows you how to use the utilities `sb2xwd` and `xwd2sb`.

Converting Starbase Format to 'xwd' Format using 'sb2xwd'

`sb2xwd` converts Starbase format window files into `xwd` format pixmaps. The pixmaps can then be printed by using `xpr` or displayed on the screen by using `xwud`, both of which are described in chapter 8.

When to Use 'sb2xwd'

Use `sb2xwd` from the command line.

Syntax and Options

`sb2xwd < filename > filename`

<*filename* The Starbase window file to be converted.

>*filename* The `xwd` pixmap file name.

Example

This example translates the Starbase window file named `mystar` into an `xwd` pixmap file named `myxwd`, then prints it on an HP LaserJet printer.

```
sb2xwd < mystar > myxwd
xpr -dev ljet myxwd | lp -oraw
```

Converting 'xwd' Format to Starbase Format using 'xwd2sb'

`xwd2sb` is the opposite of `sb2xwd`. It converts `xwd` format pixmaps into Starbase format window files.

When to Use 'xwd2sb'

Use `xwd2sb` from the command line.

Syntax and Options

`xwd2sb < filename >filename`

<*filename* The `xwd` bitmap file to be converted.

>*filename* The Starbase window file filename.

Example

This example dumps a window named `sample` into a `xwd` file called `myxwd`, translates it into a Starbase window file called `mystar`, and prints it using the Starbase `pcltrans` utility.

```
xwd -name sample -out myxwd
xwd2sb < myxwd > mystar
pcltrans mystar | lp -oraw
```

A

Using Other Window Managers

The OSF/Motif Window Manager (`mwm`) is the window manager that is described in this manual. However, two other window managers have been shipped with earlier releases of the X Window System, and are included also:

- HP Window manager (`hpwm`).
- `uwm` window manager.

If you (or your system administrator) installed X Windows on a computer that never had X Windows on it before, the OSF/Motif Window Manager will be the default window manager that is loaded and used.

If you are updating from a previous release of the X Window System, you will continue to use the window manager you had before until you deliberately convert to `mwm`.

You cannot have two window managers running at the same time on the same screen.

This appendix covers differences between the window managers and how to change between them. For specific information about each window manager, refer to the appropriate page in the Reference section.

Using 'hpwm'

The HP Window manager (hpwm) provides menus, icons, and window frames, all of which can be customized.

Window borders provide an easy way to move and resize the window without using a menu. The borders can have a 3-D look, and can also include a "system menu" button which allows you to display and select from the system menu using only a mouse. The window can be iconified or maximized by selecting other buttons on the window border.

Starting 'hpwm'

The hpwm window manager is started by a command in your `.x11start` file that looks similar to the following:

```
hpwm &        # Starts the hpwm window manager
```

If this is not in your `.x11start` file and you want to run hpwm, edit the file to include this line. Comment out the line that starts mwm by typing a pound sign (#). Your commented line will look similar to the following:

```
#mwm &     # Starts the mwm window manager
```

Differences Between 'hpwm' and 'mwm'

hpwm looks and often acts like mwm, so most of this manual is correct for it as well. The differences are mostly in the file names and resources used.

Menus

The "system menu" in hpwm is similar to the "window menu" in mwm, but it does not have accelerators and function keys. The system menu is displayed only as long as you have mouse button 1 pressed.

To display and select from the system menu:

1. Move the mouse pointer over the system menu button in the upper left corner of the window border.
2. Press *and hold down* mouse button 1.

3. Slide the pointer down the menu to the selection you want and release button 1.

There is no difference in the look or action of the root menu.

The menu pointer for **hpwm** is an ×, not an arrow as in **mwm**.

Cascading menus appear only if the pointer is moved to the right, not when the pointer is anywhere on the selection.

The menu functions that you can use to customize your own menus differ between the window managers. Refer to the man page for the specific window manager for a list of valid functions.

Icons

There is no icon box in **hpwm**.

Icon menus are slightly different in content from those listed in this manual, and are selected differently.

To display and select from an icon menu:

1. Move the mouse pointer over the icon.
2. Press *and hold down* (Shift), and press (Esc) to display the menu.
3. Use the (▲) and (▼) arrow keys to highlight the proper selection.
4. Press (Return) to make your selection.

Resources

While many of the resource names are the same in **mwm** and **hpwm**, there are some differences. This section describes the main differences. For specific information, refer to the appropriate window manager man page in the Reference Section.

If a feature does not respond when you set a resource, check that the resource name is correct for that window manager.

The default values for resources may also be different, even though the resources have the same name.

The following commonly used resources are different in **hpwm** and **mwm**:

- `font` in hpwm is `fontList` in mwm.
- `... Tile` in hpwm is `... Pixmap` in mwm.
- Resources with names referrring to the `hpwm` window menu refer to the system menu in `mwm`.
- `mwm` uses a light and dark shade of the background color to provide a 3-D look. `hpwm` uses the resource `makeColors` to do the same thing.
- Resource specifications in `mwm` can refer to a client by the client *name* as well as the client class.
- When specifying resources, `hpwm` uses a dot (`.`) between the clientname and resource, while `mwm` uses an asterisk (`*`). For example:

  `Hpwm*`*clientname.resource*: *value*
  `Mwm*`*clientname*`*`*resource*: *value*

The `hpwm` client receives configuration information from three files: `sys.Xdefaults`, `system.hpwmrc`, and `app-defaults/Hpwm`. Use these files to define resources as described elsewhere in the manual for the equivalent `mwm` files.

Using 'uwm'

Windows managed with `uwm` have variable-width borders, but do not have the functional window frame or 3-D appearance of `mwm`- or `hpwm`- managed windows.

The `uwm` window manager uses a single generic menu displayed on the root window to control the size, location, iconification, and other basic operations of the objects on the root window. `uwm` allows moving and resizing windows without using a menu.

Starting 'uwm'

The uwm window manager is started from a command in your .x11start file that looks like the following:

```
uwm &  # Starts the uwm window manager
```

If this is not in your .x11start file, edit the file to include the line. Comment out the line that starts mwm by typing a pound sign (#) at the left. The commented line should look similar to the following:

```
#mwm &  # Starts the mwm window manager
```

Configuring 'uwm'

uwm receives configuration information from the .uwmrc file. (uwm does not use the .Xdefaults or app.defaults files that the other window managers use.) If no configuration file can be found, built-in default values are used. Use this file as described for the .mwmrc file described elsewhere in this manual.

The uwm man page in the Reference Section provides a list of the variables and functions that you can use to configure uwm, along with an example startup file.

B

Reference Information

This section contains reference information about clients included with the X Window System and about the X protocol and server itself. The entries are arranged alphabetically, each starting on its own "page 1."

Table B-1. Command MAN Pages

bdftosnf(1)	xload(1)
bitmap(1)	xlsfonts(1)
gwindstop(1)	xmodmap(1)
hpterm(1)	xpr(1)
hpwm(1)	xrdb(1)
mkfontdir(1)	xrefresh(1)
mwm(1)	xseethru(1)
resize(1)	Xserver(1)
rgb(1)	xset(1)
sb2xwd(1)	xsetroot(1)
uwm(1)	xterm(1)
X(1)	xwcreate(1)
x11start(1)	xwd(1)
xclock(1)	xwd2sb(1)
xfd(1)	xwdestroy(1)
xhost(1)	xwinfo(1)
xinit(1)	xwud(1)
xinitcolormap(1)	

Table B-2. File Format MAN Pages

bdf(4)

Series 300 and 800 Only

NAME

bdftosnf - BDF to SNF font compiler for X11

SYNOPSIS

bdftosnf [-p#] [-u#] [-m] [-l] [-M] [-L] [-w] [-W] [-t] [-i] [bdf_file]

DESCRIPTION

bdftosnf reads a Bitmap Distribution Format (BDF) font from the specified file (or from standard input if no file is specified) and writes an X11 server normal font (SNF) to standard output.

OPTIONS

-p# Force the glyph padding to a specific number. The legal values are 1, 2, 4, and 8.

-u# Force the scanline unit padding to a specific number. The legal values are 1, 2, and 4.

-m Force the bit order to most significant bit first.

-l Force the bit order to least significant bit first.

-M Force the byte order to most significant bit first.

-L Force the byte order to least significant bit first.

-w Print warnings if the character bitmaps have bits set to one outside of their defined widths.

-W Print warnings for characters with an encoding of -1; the default is to silently ignore such characters.

-t Expand glyphs in "terminal-emulator" fonts to fill the bounding box.

-i Don't compute correct ink metrics for "terminal-emulator" fonts.

SEE ALSO

X(1), Xserver(1)
"Bitmap Distribution Format 2.1"

Series 300 and 800 Only

NAME

bitmap - bitmap editor for X

SYNOPSIS

bitmap [options] *filename* [*WIDTHxHEIGHT*]

DESCRIPTION

The *bitmap* program is a rudimentary tool for creating or editing rectangular images made up of 1's and 0's. Bitmaps are used in X for defining clipping regions, cursor shapes, icon shapes, and tile and stipple patterns.

USAGE

bitmap displays a grid in which each square represents a single bit in the picture being edited. Squares can be set, cleared, or inverted directly with the buttons on the pointer. A menu of higher level operations such as draw line and fill circle is provided to the side of the grid. Actual size versions of the bitmap as it would appear normally and inverted appear below the menu.

If the bitmap is to be used for defining a cursor, one of the squares in the images may be designated as the *hotspot*. This determines where the cursor is actually pointing. For cursors with sharp tips (such as arrows or fingers), this is usually at the end of the tip; for symmetric cursors (such as crosses or bullseyes), this is usually at the center.

Bitmaps are stored as small C code fragments suitable for including in applications. They provide an array of bits as well as symbolic constants giving the width, height, and hotspot (if specified) that may be used in creating cursors, icons, and tiles.

The *WIDTHxHEIGHT* argument gives the size to use when creating a new bitmap (the default is 16x16). Existing bitmaps are always edited at their current size.

If the *bitmap* window is resized by the window manager, the size of the squares in the grid will shrink or enlarge to fit.

OPTIONS

bitmap accepts the following options:

-help
This option will cause a brief description of the allowable options and parameters to be printed.

-display *display*
This option specifies the name of the X server to used.

-geometry *geometry*
This option specifies the placement and size of the bitmap window on the screen. See *X* for details.

-nodashed
This option indicates that the grid lines in the work area should not be drawn using dashed lines. Although dashed lines are prettier than solid lines, on some servers they are significantly slower.

-name *variablename*
This option specifies the variable name to be used when writing out the bitmap file. The default is to use the basename of the *filename* command line argument.

-bw *number*
This option specifies the border width in pixels of the main window.

-fn *font*
This option specifies the font to be used in the buttons.

-fg *color*
This option specifies the color to be used for the foreground.

-bg *color*
This option specifies the color to be used for the background.

-hl *color*
This option specifies the color to be used for highlighting.

-bd *color*
This option specifies the color to be used for the window border.

-ms *color*
This option specifies the color to be used for the pointer (mouse).

CHANGING GRID SQUARES

Grid squares may be set, cleared, or inverted by pointing to them and clicking one of the buttons indicated below. Multiple squares can be changed at once by holding the button down and dragging the cursor across them. Set squares are filled and represent 1's in the bitmap; clear squares are empty and represent 0's.

Button 1
This button (usually leftmost on the pointer) is used to set one or more squares. The corresponding bit or bits in the bitmap are turned on (set to 1) and the square or squares are filled.

Button 2
This button (usually in the middle) is used to invert one or more squares. The corresponding bit or bits in the bitmap are flipped (1's become 0's and 0's become 1's).

Button 3
This button (usually on the right) is used to clear one or more squares. The corresponding bit or bits in the bitmap are turned off (set to 0) and the square or squares are emptied.

MENU COMMANDS

To make defining shapes easier, *bitmap* provides 13 commands for drawing whole sections of the grid at once, 2 commands for manipulating the hotspot, and 2 commands for updating the bitmap file and exiting. A command button for each of these operations is located to the right of the grid.

Several of the commands operate on rectangular portions of the grid. These areas are selected after the command button is pressed by moving the cursor to the upper left square of the desired area, pressing a pointer button, dragging the cursor to the lower right hand corner (with the button still pressed) , and then releasing the button. The command may be aborted by pressing any other button while dragging or by releasing outside the grid.

To invoke a command, move the pointer over the command and click any button.

Clear All
This command is used to clear all of the bits in the bitmap as if Button 3 had been dragged through every square in the grid. It cannot be undone.

Set All
This command is used to set all of the bits in the bitmap as if Button 1 had been dragged through every square in the grid. It cannot be undone.

Invert All
This command is used to invert all of the bits in the bitmap as if Button 2 had been dragged through every square in the grid.

Clear Area
This command is used to clear a region of the grid as if Button 3 had been dragged through each of the squares in the region. When this command is invoked, the cursor will change shape to indicate that the area to be cleared should be selected as outlined above.

Set Area
This command is used to set a region of the grid as if Button 1 had been dragged through each of the squares in the region. When this command is invoked, the cursor will change shape to indicate that the area to be set should be selected as outlined above.

Invert Area
This command is used to inverted a region of the grid as if Button 2 had been dragged through each of the squares in the region. When this command is invoked, the cursor will change shape to indicate that the area to be inverted should be selected as outlined above.

Copy Area
This command is used to copy a region of the grid from one location to another. When this command is invoked, the cursor will change shape to indicate that the area to be copied should be selected as outlined above. The cursor should then be clicked on the square to which the upper left hand corner of the region should be copied.

Move Area
This command is used to move a region of the grid from one location to another. When this command is invoked, the cursor will change shape to indicate that the area to be moved should be selected as outlined above. The cursor should then be clicked on the square to which the upper left hand corner of the region should be moved. Any squares in the region's old position that aren't also in the new position are cleared.

Overlay Area
This command is used to copy all of the set squares in a region of the grid from one location to another. When this command is invoked, the cursor will change shape to indicate that the area to be copied should be selected as outlined above. The cursor should then be clicked on the square to which the upper left hand corner of the region should be overlaid. Only the squares that are set in the region will be touched in the new location.

Line
This command will set the squares in a line between two points. When this command is invoked, the cursor will change shape to indicate that the pointer should be clicked on the two end points of the line.

Circle
This command will set the squares on a circle specified by a center and a point on the curve. When this command is invoked, the cursor will change shape to indicate that the pointer should be clicked on the center of the circle and then over a point on the curve. Small circles may not look very round because of the size of the grid and the limits of having to work with discrete pixels.

Filled Circle
This command will set all of the squares in a circle specified by a center and a point on the curve. When this command is invoked, the cursor will change shape to indicate that the pointer should be clicked on the center of the circle and then over a point on the curve. All squares side and including the circle are set.

Flood Fill
This command will set all clear squares in an enclosed shape. When this command is invoked, the cursor will change shape to indicate that the pointer should be clicked on any empty square inside the shape to be filled. All empty squares that border horizontally or vertically with the indicated square are set out to the enclosing shape. If the shape is not closed, the entire grid will be filled.

Set HotSpot
This command designates one square in the grid as the hot spot if this bitmap to be used for defining a cursor. When the command is invoked, the cursor will change indicating that the pointer should be clicked on the square to contain the hot spot.

Clear HotSpot
This command removes any designated hot spot from the bitmap.

Write Output
This command writes a small fragment of C code representing the bitmap to the filename specified on the command line. If the file already exists, the original file will be renamed to *filename*~ before the new file is created. If an error occurs in either the renaming or the writing of the bitmap file, a dialog box will appear asking whether or not *bitmap* should use */tmp/filename* instead.

Quit
This command causes *bitmap* to display a dialog box asking whether or not it should save the bitmap (if it has changed) and then exit. Answering *yes* is the same as invoking *Write Output*; *no* causes *bitmap* to simply exit; and *cancel* will abort the *Quit* command so that more changes may be made.

FILE FORMAT

The *Write Output* command stores bitmaps as simple C program fragments that can be compiled into programs, referred to by X Toolkit pixmap resources, manipulated by other programs (see *xsetroot*), or read in using utility routines in the various programming libraries. The width and height of the bitmap as well as the hotspot, if specified, are written as preprocessor symbols at the start of the file. The bitmap image is then written out as an array of characters:

```
#define name_width 11
#define name_height 5
#define name_x_hot 5
#define name_y_hot 2

static char name_bits[] = {
   0x91, 0x04, 0xca, 0x06, 0x84,
   0x04, 0x8a, 0x04, 0x91, 0x04
};
```

The **name** prefix to the preprocessor symbols and to the bits array is constructed from the *filename* argument given on the command line. Any directories are stripped off the front of the name and any suffix beginning with a period is stripped off the end. Any remaining non-alphabetic characters are replaced with underscores. The *name_x_hot* and *name_y_hot* symbols will only be present if a hotspot has been designated using the *Set HotSpot* command.

Each character in the the array contains 8 bits from one row of the image (rows are padded out at the end to a multiple of 8 to make this is possible). Rows are written out from left to right and top to bottom. The first character of the array holds the leftmost 8 bits of top line, and the last characters holds the right most 8 bits (including padding) of the bottom line. Within each character, the leftmost bit in the bitmap is the least significant bit in the character.

This process can be demonstrated visually by splitting a row into words containing 8 bits each, reversing the bits in each word (since Arabic numbers have the significant digit on the right and images have the least significant bit on the left), and translating each word from binary to hexadecimal.

In the following example, the array of 1's and 0's on the left represents a bitmap containing 5 rows and 11 columns that spells *X11*. To its right is the same array split into 8 bit words with each row padded with 0's so that it is a multiple of 8 in length (16):

```
10001001001     10001001 00100000
01010011011     01010011 01100000
00100001001     00100001 00100000
01010001001     01010001 00100000
10001001001     10001001 00100000
```

Reversing the bits in each word of the padded, split version of the bitmap yields the left hand figure below. Interpreting each word as a hexadecimal number yields the array of numbers on the right:

```
10010001 00000100      0x91 0x04
11001010 00000110      0xca 0x06
10000100 00000100      0x84 0x04
10001010 00000100      0x8a 0x04
10010001 00000100      0x91 0x04
```

The character array can then be generated by reading each row from left to right, top to bottom:

```
static char name_bits[] = {
  0x91, 0x04, 0xca, 0x06, 0x84,
  0x04, 0x8a, 0x04, 0x91, 0x04
};
```

USING BITMAPS IN PROGRAMS

The format of *bitmap* files is designed to make bitmaps and cursors easy to use within X programs. The following code could be used to create a cursor from bitmaps defined in *this.cursor* and *this_mask.cursor*:

```
#include "this.cursor"
#include "this_mask.cursor"

XColor foreground, background;
/* fill in foreground and background color structures */
Pixmap source = XCreateBitmapFromData (display, drawable,
     this_bits, this_width, this_height);
Pixmap mask = XCreateBitmapFromData (display, drawable,
     this_mask_bits, this_mask_width, this_mask_height);
Cursor cursor = XCreatePixmapCursor (display, source, mask,
     foreground, background, this_x_hot, this_y_hot);
```

Additional routines are available for reading in *bitmap* files and returning the data in the file, in Bitmap (single-plane Pixmap for use with routines that require stipples), or full depth Pixmaps (often used for window backgrounds and borders). Applications writers should be careful to understand the difference between Bitmaps and Pixmaps so that their programs function correctly on color and monochrome displays.

For backward compatibility, *bitmap* will also accept X10 format *bitmap* files. However, when the file is written out again it will be in X11 format

X DEFAULTS

bitmap uses the following resources:

Background
The window's background color. Bits which are 0 in the bitmap are displayed in this color. This option is useful only on color displays. The default value is *white*.

BorderColor
The border color. This option is useful only on color displays. The default value is *black*.

BorderWidth
The border width. The default value is 2.

BodyFont
The text font. The default value is *variable*.

Dashed
If "off", then *bitmap* will draw the grid lines with solid lines. The default is "on".

Dashed
The line type. The default value is *true*.

Foreground
The foreground color. Bits which are 1 in the bitmap are displayed in this color. This option

is useful only on color displays. The default value is *black*.

Highlight
The highlight color. *bitmap* uses this color to show the hot spot and to indicate rectangular areas that will be affected by the *Move Area, Copy Area, Set Area,* and *Invert Area* commands. If a highlight color is not given, then *bitmap* will highlight by inverting. This option is useful only on color displays.

Mouse
The pointer (mouse) cursor's color. This option is useful only on color displays. The default value is *black*.

Geometry
The size and location of the bitmap window.

Dimensions
The *WIDTHxHEIGHT* to use when creating a new bitmap.

ENVIRONMENT
DISPLAY - the default host and display number.

SEE ALSO
X(1), *Xlib - C Language X Interface* (particularly the section on *Manipulating Bitmaps*), *XmuReadBitmapDataFromFile*

BUGS
The old command line arguments aren't consistent with other X programs.

If you move the pointer too fast while holding a pointer button down, some squares may be missed. This is caused by limitations in how frequently the X server can sample the pointer location.

There is no way to write to a file other than the one specified on the command line.

There is no way to change the size of the bitmap once the program has started.

There is no *undo* command.

COPYRIGHT

ORIGIN
bitmap by Ron Newman, MIT Project Athena; documentation by Jim Fulton, MIT X Consortium.

Series 300 and 800 Only

NAME

gwindstop - terminate the window helper facility

SYNOPSIS

gwindstop [directory] [directory] ..

DESCRIPTION

gwindstop

destroys windows and their associated pty files from named directories. The windows must have been created earlier by xwcreate(1).

directory

is the name of the directory where the pty files for the windows reside. If **directory** name is not supplied, */dev/screen* is taken to be the desired directory. Otherwise, if the directory argument implies an absolute pathname, then it will be taken to be the desired directory. Otherwise, the directory name will be taken to be relative to the value of the environment variable $WMDIR. If $WMDIR is not defined in the environment, the directory name will be taken to be relative to */dev/screen.* Note: if $WMDIR is defined in the environment, it must represent an absolute pathname.

DIAGNOSTICS

If the windows in the indicated directory are successfully destroyed, then the program remains silent. If one or more directories could not be found, an error message ("Invalid directory") will be printed on the standard output.

ORIGIN

Hewlett-Packard Company

SEE ALSO

xwcreate(1), xwdestroy(1).

Series 300 and 800 Only

NAME

hpterm - X window system Hewlett-Packard terminal emulator

SYNOPSIS

hpterm [-toolkitoption] [-option]

DESCRIPTION

The *hpterm* program is a terminal emulator for the X Window system. It provides a Term0 compatible terminal for programs that can't use the window system directly. It also emulates many of the blockmode features of HP terminals. To use these features, set the termId resource as explained below.

OPTIONS

The *hpterm* terminal emulator accepts all of the standard X Toolkit command line options along with additional options all of which are listed below (if the option begins with a '+' instead of a '-', the option is restored to its default value):

-b *number*
This option specifies the size of the inner border (the distance between the outer edge of the character and the window border) in pixels. Associated resource: "***borderWidth**."

-background *color*
This option specifies the color to use for the background of the window. Associated resource: "***background**."

-bd *color*
This option specifies the color to use for the border of the window. Associated resource: "***borderColor**."

-bg *color*
This option specifies the color to use for the background of the window. Associated resource: "***background**."

-borderwidth *number*
This option specifies the width in pixels of the border surrounding the window. Associated resource: "***TopLevelShell.borderWidth**."

-bw *number*
This option specifies the width in pixels of the border surrounding the window. Associated resource: "***TopLevelShell.borderWidth**."

-cr *color* This option specifies the color to use for the text cursor. Associated resource: "***cursorColor**."

-display *display*
This option specifies the X server to contact; see *X*(1). Associated resource: none.

-e *command* [*arguments ...*]
This option specifies the command (and its command line arguments) to be run in the *hpterm* window. The default is to start the user's shell. **This must be the last option on the command line.** Associated resource: none.

-fb *font* This option specifies a font to be used when displaying bold (alternate) text. This font must be the same height and width as the normal (primary) font. If only one of the normal (primary) or bold (alternate) fonts is specified, it will be used for both fonts. Refer to the NLS section. Associated resource: "***boldFont**."

-fg *color* This option specifies the color to use for displaying text. Associated resource: "***foreground**."

-fn *font* This option specifies a font to be used when displaying normal (primary) text. If only one of the normal (primary) or bold (alternate) fonts is specified, it will be used for both fonts. Refer to the NLS section. Associated resource: "***font**."

-font *font*
This option specifies a font to be used when displaying normal (primary) text. If only one of the normal (primary) or bold (alternate) fonts is specified, it will be used for both

fonts. Associated resource: "***font.**"

-foreground *color*
This option specifies the color to use for displaying text. Associated resource: "***foreground.**"

-geometry *geometry*
This option specifies the preferred size and position of the *hpterm* window; see *X*(1). Associated resource: "***term0.geometry.**"

-help This option will display a help message. Associated resource: none.

-i This option indicates that *hpterm* should supply the window manager with a bitmapped icon. Associated resource: "**bitmapIcon.**"

+i This option indicates that the window manager should generate its own icon for *hpterm*. Associated resource: "**bitmapIcon.**"

-iconic This option indicates that *hpterm* should be placed on the display in icon form. Associated resource: "***term0.iconic.**"

+iconic This option indicates that *hpterm* should not be placed on the display in icon form. Associated resource: "***term0.iconic.**"

-kshmode
This option indicates that *hpterm* should convert characters entered with the extend key pressed into a two character sequence consisting of an ASCII escape followed by the un-extended character. Associated resource: "***kshMode.**"

-l This option indicates that *hpterm* should send all terminal output to a log file as well as to the screen. Associated resource: "***logging.**"

+l This option indicates that *hpterm* should not do logging. Associated resource: "***logging.**"

-lf *file* This option specifies the name of the file to which the output log described above is written. If *file* begins with a pipe symbol (|), the rest of the string is assumed to be a command to be used as the endpoint of a pipe. The default filename is "**HptermLog***XXXXX*" (where *XXXXX* is the process id of *hpterm*) and is created in the directory from which *hpterm* was started (or the user's home directory in the case of a login window). Associated resource: "***logFile.**"

-ls This option indicates that the shell that is started in the *hpterm* window should be a login shell (i.e. the first character of argv[0] will be a dash, indicating to the shell that it should read the user's .login or .profile). Associated resource: "***loginShell.**"

+ls This option indicates that the shell that is started should not be a login shell (i.e. it will be a normal "subshell"). Associated resource: "***loginShell.**"

-mb This option indicates that the pointer cursor should be put into blanking mode. In this mode, the cursor will turn on when the pointer is moved, and will be blanked either after a selectable number of seconds or after keyboard input has occurred. The delay is set via the "**pointerBlankDelay**" resource. Associated resource: "***pointerBlank.**"

+mb This option indicates that the pointer cursor should remain on. Associated resource: "***pointerBlank.**"

-mc *mode*
This option determines how hpterm will generate the foreground color, shadow colors, and shadow tiles of the scrollbar and softkey widgets. Valid modes are "all", "shadow", and "none." Associated resource: "***makeColors.**"

-ms *color*
This option specifies the color to be used for the pointer cursor. Associated resource: "***pointerColor.**"

-name *name*
This option specifies the application name under which resources are to be obtained, rather than the default executable file name ("hpterm"). Associated resource: "**.name.**"

-reverse This option indicates that reverse video should be simulated by swapping the foreground and background colors. Associated resource: "***reverseVideo.**"

-rv This option indicates that reverse video should be simulated by swapping the foreground and background colors. Associated resource: "***reverseVideo.**"

+rv This option indicates that reverse video should not be simulated. Associated resource: "***reverseVideo.**"

-sb This option indicates that a scrollbar should be displayed. Associated resource: "***scrollBar.**"

+sb This option indicates that a scrollbar should not be displayed. Associated resource: "***scrollBar.**"

-sbbg *color*
This option specifies the color to use for the background of the scrollbar window. Associated resource: "***scrollBar.background.**"

-sbfg *color*
This option specifies the color to use for the foreground of the scrollbar window. This value will be ignored if the **makeColors** resource is set to "**all.**" Associated resource: "***scrollBar.foreground.**"

-skbg *color*
This option specifies the color to use for the background of the softkey window. Associated resource: "***softkey.background.**"

-skfg *color*
This option specifies the color to use for displaying softkey text. This value will be ignored if the **makeColors** resource is set to "**all.**" Associated resource: "***softkey.foreground.**"

-skfn *font*
This option specifies a font to be used when displaying softkey text. Associated resource: "***softkey.font.**"

-sl *number*[*suffix*]
This option indicates the number of off screen lines to be saved in the terminal buffer. If no suffix is included or the suffix is "**l**" the total length of the terminal buffer will be *number* plus the length of the terminal window. If the suffix is "**s**" the total length of the terminal buffer will be (*number* plus one) times the length of the terminal window. Associated resource: "***saveLines.**"

-ti *name* This option specifies a name for hpterm to use when identifying itself to application programs. To turn on blockmode, set this resource to "2392A". Associated resource: "***termId.**"

-title *name*
This option specifies a window title for hpterm. This string may be used by the window manager when displaying the application. Associated resource: "**.TopLevelShell.title.**"

-tn *name*
This option specifies a name for hpterm to set the "**$TERM**" environment variable to. Associated resource: "***termName.**"

-vb This option indicates that a visual bell is preferred over an audible one. Instead of ringing the terminal bell whenever a Control-G is received, the window will be flashed. Associated resource: "***visualBell.**"

+vb This option indicates that a visual bell should not be used. Associated resource: "***visualBell.**"

-xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options. Associated resource: none.

-C This option indicates that the window should receive console output. Associated resource: none.

-L This option indicates that *hpterm* was started by *init*(1m). In this mode, *hpterm* does not try to allocate a new pseudoterminal as *init*(1m) has already done so. In addition, the system program *getty*(1m) is run instead of the user's shell. This option requires a pty name as a separate last argument. **This option should never be used by users when starting terminal windows.** Associated resource: none.

-S*ccn* This option specifies the last two letters of the name of a pseudoterminal to use in slave mode, and the file descriptor of the pseudoterminal's master. This allows *hpterm* to be used as an input and output channel for an existing program and is sometimes used in specialized applications such as *pam*(1). Associated resource: none.

The following command line arguments are provided for compatibility with older versions. They may not be supported in future releases as the X Toolkit provides standard options that accomplish the same task.

=geometry
This option specifies the preferred size and position of the *hpterm* window; see *X*(1). It is equivalent to "**-geometry** *geometry*." Associated resource: "***term0.geometry**."

#geometry
This option specifies the preferred position of the icon window. It is shorthand for specifying the "***iconGeometry**" resource. Associated resource: "**.iconGeometry**."

-T *string* This option specifies the title for *hpterm*'s window. It is equivalent to "**-title** *string*." Associated resource: "**.TopLevelShell.title**."

-n *string* This option specifies the icon name for *hpterm*'s windows. It is shorthand for specifying the "***iconName**" resource. Associated resource: "***iconName**."

-r This option indicates that reverse video should be simulated by swapping the foreground and background colors. It is equivalent to "**-reversevideo**" or "**-rv**." Associated resource: "***reverseVideo**."

+r This option indicates that reverse video should no be simulated. It is equivalent to "**+rv**." Associated resource: "***reverseVideo**."

-w *number*
This option specifies the width in pixels of the border surrounding the window. It is equivalent to "**-borderwidth** *number*" or "**-bw** *number*." Associated resource: "***TopLevelShell.borderWidth**."

RESOURCES

The *hpterm* window consists of an X Toolkit shell widget which contains a term0 widget. The term0 widget contains a scrollbar widget and a softkey widget. Resources specific to the shell widget are:

hpterm Resource Set			
Name	**Class**	**Type**	**Default**
borderColor	BorderColor	Pixel	black
borderWidth	BorderWidth	int	2
geometry	Geometry	string	
iconGeometry	IconGeometry	string	
name	Name	string	hpterm
title	Title	string	Terminal emulator

borderColor
This resource defines the border color of the *hpterm* window.

borderWidth
This resource specifies the width of the *hpterm* window border. This value may be modified by the window manager.

geometry
This resource specifies the preferred size and position of the *hpterm* window.

iconGeometry
This resource specifies the preferred size and position of *hpterm* when iconified. It is not necessarily obeyed by all window managers.

name This resource specifies the name of the instance of the program. It is used when extracting resources from the resource database.

title This resource specifies the window title for *hpterm*. This string may be used by the window manager when displaying this application.

term0 Resource Set			
Name	**Class**	**Type**	**Default**
allowSendEvents	AllowSendEvents	Boolean	FALSE
background	Background	Pixel	white
bitmap	Bitmap	string	none
bitmapIcon	BitmapIcon	Boolean	FALSE
boldFont	Font	string	see NLS below
copyLine	CopyLine	string	shift right
cursorColor	Foreground	Pixel	black
cut	Cut	string	shift left
flashBorder	FlashBorder	Boolean	FALSE
font	Font	string	see NLS below
foreground	Foreground	Pixel	black
halfBrightInhibit	HalfBrightInhibit	Boolean	FALSE
iconic	Iconic	Boolean	FALSE
internalBorder	BorderWidth	int	2
keyboardLanguage	KeyboardLanguage	string	see NLS below
keyboardLanguageList	KeyboardLanguageList	string	see NLS below
kshMode	KshMode	Boolean	FALSE
logFile	LogFile	string	HptermLog*XXXXX*
logging	Logging	Boolean	FALSE
loginShell	LoginShell	Boolean	FALSE
makeColors	MakeColors	string	none
paste	Paste	string	shift middle
pointerBlank	PointerBlank	Boolean	FALSE
pointerBlankDelay	PointerBlankDelay	int	3
pointerColor	Foreground	Pixel	black
pointerShape	PointerShape	string	xterm
reverseVideo	ReverseVideo	Boolean	FALSE
saveLines	SaveLines	string	1s
scrollBar	ScrollBar	Boolean	FALSE
softkeySelect	SoftkeySelect	string	left
stickyNextCursor	StickyCursor	Boolean	TRUE
stickyPrevCursor	StickyCursor	Boolean	TRUE
termId	TermId	string	X-hpterm
termName	TermName	string	hpterm
visualBell	VisualBell	Boolean	FALSE

term0.*fontLanguage* (class Term0.FontLanguage) Resource Set			
Name	**Class**	**Type**	**Default**
primary.high	FontPosition.Size	string	see NLS below
primary.medium	FontPosition.Size	string	see NLS below
primary.low	FontPosition.Size	string	see NLS below
alternate.high	FontPosition.Size	string	see NLS below
alternate.medium	FontPosition.Size	string	see NLS below
alternate.low	FontPosition.Size	string	see NLS below

allowSendEvents
This resource defines whether or not synthetic key and button events (generated using the X protocol SendEvent request) should be interpreted or discarded.

background
This resource defines the background color of the text window.

bitmap This resource defines whether or not *hpterm* will override its built in bitmap icon with a user specified bitmap icon. If the path does not begin with a "/" or "./", it will be processed relative to "/usr/lib/X11/bitmaps".

bitmapIcon
This resource defines whether or not *hpterm* will supply the window manager with a bitmapped icon. The supplied bitmap may be ignored by the window manager.

boldFont
This resource defines the font used for bold (alternate) text. See "NLS" below for defaults.

copyLine
This resource defines the pointer button/modifier combination to be used to activate the CopyLine function. See "POINTER USAGE" below.

cut This resource defines the pointer button/modifier combination to be used to activate the Cut function. See "POINTER USAGE" below.

cursorColor
This resource defines the text cursor color. The pointer cursor color is defined by the **pointerColor** resource.

flashBorder
This resource defines whether or not *hpterm* window border will change color when the pointer cursor enters or leaves the window.

font This resource defines the font used for normal (primary) text. See "NLS" below for defaults.

fontLanguage.**primary.high**
This resource defines the default normal (primary) font for displays with high resolution monitors. See "NLS" below for additional information.

fontLanguage.**primary.medium**
This resource defines the default normal (primary) font for displays with medium resolution monitors. See "NLS" below for additional information.

fontLanguage.**primary.low**
This resource defines the default normal (primary) font for displays with low resolution monitors. See "NLS" below for additional information.

fontLanguage.**alternate.high**
This resource defines the default bold (alternate) font for displays with high resolution monitors. See "NLS" below for additional information.

fontLanguage.**alternate.medium**
This resource defines the default bold (alternate) font for displays with medium resolution monitors. See "NLS" below for additional information.

fontLanguage.**alternate.low**
This resource defines the default bold (alternate) font for displays with low resolution monitors. See "NLS" below for additional information.

foreground
This resource defines the foreground (text) color of the text window.

halfBrightInhibit
This resource defines whether or not half-bright enhancements will be not be generated. When true, full-bright characters will be used instead of half-bright characters.

iconic This resource defines whether or not *hpterm* will start up in iconic form.

internalBorder
This resource defines the number of pixels between the characters and the window border.

keyboardLanguage
This resource defines the default keyboard language *hpterm* should use. See "NLS" below for details and defaults.

keyboardLanguageList
This resource defines the list of keyboard languages that may be selected from the terminal configuration menu. See "NLS" below for detail and defaults.

kshMode
This resource defines whether or not *hpterm* will operate in ksh mode. In ksh mode *hpterm* will should convert characters entered with the extend key pressed into a two character sequence consisting of an ASCII escape followed by the un-extended character.

logFile This resource defines the name of the file to which a terminal session is logged. The default is "**HptermLog***XXXXX*" (where *XXXXX* is the process id of *hpterm*).

logging This resource defines whether or not a terminal session will be logged. It is also available at runtime via the Device Control menu.

loginShell
This resource defines whether or not the shell to be run in the window will be started as a login shell (i.e., the first character of argv[0] will be a dash, indicating to the shell that it should read the user's .login or .profile).

makeColors
This resource defines how the **bottomShadowColor**, **foreground**, and **topShadowColor** resources of the scrollbar and softkey widgets will be generated. If the value of this resource is "all" then *hpterm* will use the value of the **foreground** resource of the softkey and scrollbar widgets to generate values for the **bottomShadowColor**, **foreground**, and **topShadowColor** resources such that there is a 3-D look. In this case the **topShadowTile** and **bottomShadowTile** are always set to "foreground." If the **makeColors** resource value is "shadow" the **bottomShadowColor** and **topShadowColor** will be generated but **foreground** will not be generated. If the **makeColors** resource value is set to "none" then no colors will be generated.

paste This resource defines the pointer button/modifier combination to be used to activate the Paste function. See "POINTER USAGE" below.

pointerBlank
This resource defines whether or not *hpterm* will put the pointer cursor into blanking mode. In blanking mode, the pointer cursor will turn on when the pointer is moved, and will be blanked either after a selectable number of seconds or after keyboard input has occurred. The delay is set via the **pointerBlankDelay** resource.

pointerBlankDelay
This resource defines the number of seconds to wait before blanking the pointer cursor after the pointer has been moved. When set to "0", the pointer will be blanked only upon keyboard input.

pointerColor
This resource defines the pointer cursor color. The text cursor color is defined by the **cursorColor** resource.

pointerShape
This resource defines the pointer cursor shape. Valid cursor shapes may be found in the file "/usr/lib/X11/cursorfont.h." Shapes are specified as the name with the leading "XC_" dropped. Valid cursor shapes include "left_ptr", "crosshair", and "xterm."

reverseVideo
This resource defines whether or not reverse video will be simulated by swapping the

foreground and background colors.

saveLines
This resource defines the number of lines in the terminal buffer beyond the length of the window. The resource value consists of a "*number*" followed by an optional "*suffix*." If no *suffix* is included or the *suffix* is "l" the total length of the terminal buffer will be *number* plus the length of the terminal window. If the suffix is "s" the total length of the terminal buffer will be (*number* plus one) times the length of the terminal window. *Hpterm* **will try to maintain the same buffer to window ratio when the window is resized larger.**

scrollBar
This resource defines whether or not the scrollbar will be displayed.

softkeySelect
This resource defines the pointer button/modifier combination to be used for selecting softkeys. See "POINTER USAGE" below.

stickyNextCursor
This resource defines whether or not the cursor should be homed when the **Next** key is pressed. When true, the cursor will be in the same screen position after the key is pressed that it was in before pressing the key. When false, the cursor will be moved to the upper left hand corner of the screen after the key is pressed.

stickyPrevCursor
This resource defines whether or not the cursor should be homed when the **Prev** key is pressed. When true, the cursor will be in the same screen position after the key is pressed that it was in before pressing the key. When false, the cursor will be moved to the upper left hand corner of the screen after the key is pressed.

termId This resource defines the name for *hpterm* to use when identifying itself to application programs.

termName
This resource defines the string for set the "**$TERM**" environment variable.

visualBell
This resource defines whether or not a visible bell (i.e. flashing) should be used instead of an audible bell when Control-G is received.

The following resources are specified as part of the "softkey" widget (name "softkey", class "Softkey"). For example, the softkey font resource would be specified one of:

```
HPterm*softkey*font:    hp8.8x16
HPterm*Softkey*font:    hp8.8x16
*Softkey*Font:          hp8.8x16
```

Additional resources and information can be found in the **XwPrimitive**(3X) and **CORE**(3X) man pages along with additional information about the various shadow options.

Softkey Resource Set			
Name	**Class**	**Type**	**Default**
background	Background	Pixel	white
bottomShadowColor	Foreground	Pixel	black (see below)
bottomShadowTile	BottomShadowTile	string	foreground (see below)
font	Font	string	(see below)
foreground	Foreground	Pixel	black (see below)
topShadowColor	Background	Pixel	white (see below)
topShadowTile	TopShadowTile	string	50_foreground (see below)

background
This resource defines the background color of the softkey window.

bottomShadowColor
This resource defines the color that is combined with the bottom shadow tile and foreground color to create a pixmap used to draw the bottom and right sides of the softkey borders. This may be overridden by the term0 **makeColors** resource described above.

bottomShadowTile
This resource defines the tile used in creating the pixmap used for drawing the bottom and right shadows for the softkey borders. Valid tile names are described in **XwCreateTile**(3X). This may be overridden by the term0 **makeColors** resource described above.

font This resource defines the font used for softkey text. The softkey font will default to the normal (primary) font of the text window.

foreground
This resource defines the foreground (text) color of the softkey window. This may be overridden by the term0 **makeColors** resource described above.

topShadowColor
This resource defines the color that is combined with the top shadow tile and foreground color to create a pixmap used to draw the top and left sides of the softkey borders. This may be overridden by the term0 **makeColors** resource described above.

topShadowTile
This resource defines the tile used in creating the pixmap used for drawing the top and left shadows for the softkey borders. Valid tile names are described in **XwCreateTile**(3X). This may be overridden by the term0 **makeColors** resource described above.

The following resources are specified as part of the "Xwscrollbar" widget (name "scrollBar", class "ScrollBar"). Some example scrollbar resources are:

```
HPterm*scrollBar*initialDelay:    10
HPterm*ScrollBar*RepeatRate:      10
*ScrollBar*Granularity:           1
hpterm*scrollBar*width:           20
```

Additional resources and information can be found in the **XwPrimitive**(3X), **XwScrollBar**(3X), **XwValuator**(3X), and **Core**(3X) man pages along with additional information about the various shadow options.

Scrollbar Resource Set (name "scrollBar", class "ScrollBar")			
Name	**Class**	**Type**	**Default**
background	Background	Pixel	white
bottomShadowColor	Foreground	Pixel	black (see below)
bottomShadowTile	BottomShadowTile	string	foreground (see below)
foreground	Foreground	Pixel	black (see below)
granularity	Granularity	int	2
initialDelay	InitialDelay	int	500
repeatRate	RepeatRate	int	100
topShadowColor	Background	Pixel	white (see below)
topShadowTile	TopShadowTile	string	50_foreground (see below)
width	Width	int	10

background
This resource defines the background color of the scrollbar window.

bottomShadowColor
This resource defines the color that is combined with the bottom shadow tile and foreground color to create a pixmap used to draw the bottom and right sides of the scrollbar borders. This may be overridden by the term0 **makeColors** resource described above.

bottomShadowTile
This resource defines the tile used in creating the pixmap used for drawing the bottom and right shadows for the scrollbar borders. Valid tile names are described in **XwCreateTile**(3X). This may be overridden by the term0 **makeColors** resource described above.

foreground
This resource defines the foreground color of the scrollbar window. This may be overridden by the term0 **makeColors** resource described above.

granularity
This resource defines the number of lines to advance the slider when the button is being held down on an arrow. The value is defined in milliseconds.

initialDelay
This resource defines the delay to wait between the time the button is held down on an arrow before the slider starts its repetitive movement. The value is defined in milliseconds.

repeatRate
This resource defines the continuous repeat rate to use to move the slider while the button is being held down on an arrow. The value is also defined in milliseconds.

topShadowColor
This resource defines the color that is combined with the top shadow tile and foreground color to create a pixmap used to draw the top and left sides of the scrollbar borders. This may be overridden by the term0 **makeColors** resource described above.

topShadowTile
This resource defines the tile used in creating the pixmap used for drawing the top and left shadows for the scrollbar borders. Valid tile names are described in **XwCreateTile**(3X). This may be overridden by the term0 **makeColors** resource described above.

width This resource defines the width of the scrollbar in pixels.

POINTER USAGE
Hpterm allows you to cut and paste text within its own or other windows. All cutting and pasting is

done to/from the first global cut buffer.

The default button assignments may be changed via various resource strings. The default button functions are all activated when the "shift" key is pressed. The cut and paste functions and their default button assignments are:

CopyLine
The left hand button "cuts" the text from the pointer (at button release) through the end of line (including the new line), saving it in the cut buffer, and immediately "pastes" the line, inserting it as keyboard input. This provides a history mechanism.

Cut The center button is used to "cut" text into the cut buffer. Move the pointer to the beginning of the text to cut, press the button, move the cursor to the end of the region, and release the button. The "cut" text will not include the character currently under the pointer.

Paste The right hand button "pastes" the text from the cut buffer, inserting it as keyboard input.

The copyLine, cut, and paste key functions can be configured to any button and modifier combination desired via various resources. Each assignment consists of an optional combination of modifiers ("none" or any combination of "shift", "meta", "lock", "control", "mod1", ..., "mod5" separated by blanks), followed by a "|" and the name of the button ("left", "middle", "right", "button1", ..., "button5"). For example, if it is desired for the cut function to be associated with the middle button with shift and control pressed, one could use the following resource line:

```
*cut:                shift control | middle
```

For a full list of resource names, see "RESOURCES" above.

NLS

Hpterm currently supports 23 different language versions of the HP keyboard. It is possible to switch between different languages via the "terminal configuration" menu. A list of language to choose from along with their order is specified via the "keyboardLanguageList" resource. The "keyboardLanguageList" resource consists of a list of keyboard languages separated by spaces, tabs, or new lines. Valid keyboard languages may be found in the file "/usr/lib/X11/XHPlib.h." Keyboard languages are specified as the language with the leading "**KB_**" dropped. The default value for the "keyboardLanguageList" resource is "US_English Belgian Canada_English Danish Dutch Finnish French Canada_French Swiss_French German Swiss_German Italian Norwegian Euro_Spanish Latin_Spanish Swedish UK_English Katakana Swiss_French2 Swiss_German2 Japanese Korean S_Chinese T_Chinese."

The initial keyboard language is specified via the "keyboardLanguage" resource. If the string is NULL, the language of the server's keyboard will be used. If the keyboard language specified is not included in the keyboardLanguageList resource, the first language included in the keyboardLanguageList resource will be used. The default is to use the language of the server's keyboard.

Hpterm will try to select default fonts which match your monitor and your keyboard language. If the normal (primary) and bold (alternate) fonts are specified, they will be used. If only one is specified (via either command line options or resources), it will be used for both the normal (primary) and bold (alternate) fonts. If neither normal (primary) or bold (alternate) fonts are specified, *hpterm* trys to find them based on the default keyboard language. The default keyboard language is indicated on the "terminal configuration" menu. The built in defaults may be overridden via the "term0.*fontLanguage*" resources. *FontLanguage* varies with the keyboard language as follows.

keyboard language	*fontLanguage*
Katakana	hp_kana8
Japanese	hp_japanese
Korean	hp_korean
T_Chinese	hp_chinese_t
S_Chinese	hp_chinese_s
all others HP keyboards	hp_roman8
non HP keyboards	iso_8859_1

The default font size used will depend upon the resolution of the monitor as follows:

monitor resolution	font size
72 DPI or less	low
greater than 72 DPI and less 100 DPI	medium
100 DPI or greater	high

For example, resource specifications for the US_English, German, and Finnish keyboards would be:

```
HPterm*hp_roman8.primary.high:      *courier-medium-r-normal--14*hproman8*
HPterm*hp_roman8.primary.medium:    *courier-medium-r-normal--12*hproman8*
HPterm*hp_roman8.primary.low:       *courier-medium-r-normal--8*hproman8*
HPterm*hp_roman8.alternate.high:    *courier-bold-r-normal--14*hproman8*
HPterm*hp_roman8.alternate.medium:  *courier-bold-r-normal--12*hproman8*
HPterm*hp_roman8.alternate.low:     *courier-bold-r-normal--8*hproman8*
```

For the Japanese keyboard, resource specifications would be:

```
HPterm*hp_japanese.primary.high:      jpn.8x18
HPterm*hp_japanese.primary.medium:    jpn.8x18
HPterm*hp_japanese.primary.low:       jpn.8x18
HPterm*hp_japanese.alternate.high:    math.8x18
HPterm*hp_japanese.alternate.medium:  math.8x18
HPterm*hp_japanese.alternate.low:     math.8x18
```

If these fonts can not be found, the font "fixed" will be used for both the normal (primary) and bold (alternate) fonts. These resources are for font defaults only and will be ignored if either the normal (primary) or bold (alternate) fonts are specified.

Control-N will switch to the bold (alternate) font and control-O will switch back to the normal (primary) font. *Hpterm* will switch back to the normal (primary) font automatically at the beginning of each line.

ENVIRONMENT

Hpterm sets the environment variable "**$TERM**" properly for the size window you have created. It sets "**$LINES**" and "**$COLUMNS**" to be the number of lines and columns of the terminal screen. It also uses and sets the environment variable "**$DISPLAY**" to specify its server connection. The *resize*(1) command may be used to reset "**$LINES**" and "**$COLUMNS**" after the window size has been changed.

ORIGIN

Hewlett-Packard Company

SEE ALSO

X(1), resize(1), xset(1), xterm(1), pty(4), Core(3X), XwScrollBar(3X), XwPrimitive(3X), XwCreateTile(3X), XwValuator(3X), XwArrow(3X)

NAME

hpwm - the Hewlett Packard window manager for X

SYNOPSIS

hpwm [options]

DESCRIPTION

The Hewlett-Packard Window Manager (*hpwm*) is an X11 client that provides window management functionality and some session management functionality. It provides functions that facilitate control (by the user and the programmer) of elements of window state such as placement, size, icon/normal display, input focus ownership etc. It also provides session management functions such as stopping a client.

When *hpwm* is invoked, it retrieves configuration resource values from the following files.

/usr/lib/X11/app-defaults/Hpwm
.Xdefaults
hpwm resource description file (.hpwmrc)

OPTIONS

-display *display*

This option specifies the display to use; see *X(1)*.

-xrm *resourcestring*

This option specifies a resource string to use.

X DEFAULTS

Hpwm is configured from its resource database. This database is built from the resource specifications in the *.Xdefaults* and */usr/lib/X11/app-defaults/Hpwm* resource files. Entries in these files may refer to other resource files that specify specific types of resources (e.g., bitmaps and menus). If the same resource is specified in more than one of the resource files the resource specification in the *.Xdefaults* file has precedence over the specification in the */usr/lib/X11/app-defaults/Hpwm* file. Hpwm has built-in default values for all the resources that it uses (refer to the descriptions of specific resources).

Hpwm is the resource class name of hpwm and **hpwm** is the resource name used by hpwm to look up resources. In the following discussion of resource specification "Hpwm" and "hpwm" can be used interchangeably.

Hpwm uses the following types of resources:

General Appearance Resources:

These resources are used to specify appearance attributes of window manager user interface components. They can be applied to the appearance of window manager menus, client window frames and icons.

Specific Appearance And Behavior Resources:

These resources are used to specify hpwm appearance and behavior (e.g., window management policies). They are not set separately for different hpwm user interface components.

Client Class Specific Resources:

These hpwm resources can be set for a particular class of client windows. They specify class-specific icon and client window frame appearance and behavior.

Resource identifiers can be either a resource name (e.g., "foreground") or a resource class (e.g., "Foreground"). If the value of a resource is a filename and if the filename is prefixed by "~/" then it is relative to the path contained in the *$HOME* environment variable (generally the user's home directory).

General Appearance Resources

The syntax for specifying *general appearance resources* that apply to window manager icons, menus

and client window frames is:

"**Hpwm***<resource_id>"

For example, "Hpwm*foreground" is used to specify the foreground color for hpwm menus, icons, client window frames.

The syntax for specifying *general appearance resources* that apply to a particular hpwm component is:

"**Hpwm*[menu|icon|client]***<resource_id>"

If *menu* is specified the resource is applied only to hpwm menus, if *icon* is specified the resource is applied to icons, and if *client* is specified the resource is applied to client window frames. For example, "Hpwm*icon*foreground" is used to specify the foreground color for hpwm icons, "Hpwm*menu*foreground" specifies the foreground color for hpwm menus, and "Hpwm*client*foreground" is used to specify the foreground color for hpwm client window frames.

The following *general appearance resources* can be specified:

activeBackground (class **Background**)
Specifies the background color of the hpwm decoration when the window is active (has the keyboard focus). This resource can have any legal color as a value. The default value is the *background* general appearance resource value.

activeBackgroundTile (class **ActiveBackgroundTile**)
This resource specifies the background tile of the hpwm decoration when the window is active (has the keyboard focus). This resource can be any legal HP X Widget tile value (See *XwCreateTile*(*3X*)). The default value is "background".

activeBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the hpwm decoration when the window is active (has the keyboard focus). The default value is the *bottomShadowColor* general appearance resource value.

activeBottomShadowTile (class **BottomShadowTile**)
This resource specifies the bottom shadow tile of the hpwm decoration when the window is active (has the keyboard focus). The default value is the *bottomShadowTile* general appearance resource value.

activeForeground (class **Foreground**)
This resource specifies the foreground color of the hpwm decoration when the window is active (has the keyboard focus). This resource can have any legal color as a value. The default value is the *foreground* general appearance resource value.

activeTopShadowColor (class **Background**)
This resource specifies the top shadow color of the hpwm decoration when the window is active (has the keyboard focus). The default value is the *topShadowColor* general appearance resource value.

activeTopShadowTile (class **TopShadowTile**)
This resource specifies the top shadow tile of the hpwm decoration when the window is active (has the keyboard focus). The default value is the *topShadowTile* general appearance resource value.

background (class **Background**)
This resource specifies the background color. Any legal X color may be specified. The default is "White".

backgroundTile (class **BackgroundTile**)
This resource specifies the background tile of the hpwm decoration when the window is inactive (does not have the keyboard focus). This resource can be any legal HP X Widget tile value. The default value is "25_foreground".

bottomShadowColor (class **Foreground**)
This resource specifies the top shadow color. This color is used for the lower and right

bevels of the window manager decoration. Any legal X color may be specified. The default is "Black".

bottomShadowTile (class **BottomShadowTile**)
This resource specifies the bottom shadow tile. This tile is used for the lower and right bevels of the window manager decoration. Any legal HP X Widget tile may be specified. The default is "foreground".

font (class **Font**)
This resource specifies the font used in menus, window titles, and icon labels. Any available X font may be specified. The character encoding of the font should match the character encoding of the strings that are used. The default is "fixed."

foreground (class **Foreground**)
This resource specifies the foreground color. Any legal X color may be specified. The default is "Black".

makeActiveColors (class **MakeColors**)
If the value of this resource is "all" (or "true") then hpwm will use the value of the **activeBackground** resource to make values for the **activeBottomShadowColor**, **activeForeground** and **activeTopShadowColor** resources that provide a 3-D appearance. In this case the **activeTopShadowTile** and **activeBottomShadowTile** are always set to "foreground". If the makeColors resource value is "shadow" the top and bottom shadow colors will be made but the foreground color will not be made. If the makeColors resource value is "none" (or "false") then no colors will be automatically made. The default value for this resource is "shadow".

makeColors (class **MakeColors**)
If the value of this resource is "all" then hpwm will use the value of the **background** resource to make values for the **bottomShadowColor**, **foreground** and **topShadowColor** resources that provide a 3-D appearance. In this case the **topShadowTile** and **bottomShadowTile** are always set to "foreground". If the makeColors resource value is "shadow" the top and bottom shadow colors will be made but the foreground color will not be made. If the makeColors resource value is "none" then no colors will be automatically made. The default value for this resource is "shadow".

topShadowColor (class **Background**)
This resource specifies the top shadow color. This color is used for the upper and left bevels of the window manager decoration. Any legal X color may be specified. The default is "White".

topShadowTile (class **TopShadowTile**)
This resource specifies the top shadow tile. This tile is used for the upper and left bevels of the window manager decoration. Any legal HP X Widget tile may be specified. The default is "50_foreground".

Specific Appearance And Behavior Resources

The syntax for specifying *specific appearance and behavior resources* is:

"**Hpwm***<resource_id>"

For example, "Hpwm*keyboardFocusPolicy" is used to specify the window manager policy for setting the keyboard focus to a particular client window.

The following *specific appearance and behavior resources* can be specified:

bitmapDirectory (class **BitmapDirectory**)
This resource identifies a directory that is to be searched for bitmaps that are referenced by hpwm resources. This directory is searched if a bitmap is specified without an absolute pathname. The default value for this resource is "/usr/include/X11/bitmaps".

buttonBindings (class **ButtonBindings**)
This resource identifies the set of button bindings for window management functions. The named set of button bindings is specified in the *hpwm resource description file* file.

The default value for this resource is "DefaultButtonBindings".

colormapFocusPolicy (class **ColormapFocusPolicy**)
This resource indicates the colormap focus policy that is to be used. If the resource value is "explicit" then a colormap selection action is done on a client window to set the colormap focus to that window. If the value is "pointer" then the client window that contains the pointer will have the colormap focus. If the value is "keyboard" then the client window that has the keyboard input focus will have the colormap focus. The default value for this resource is "keyboard".

configFile (class **ConfigFile**)
The resource value is the pathname for an *hpwm resource description file*. The default is **.hpwmrc** in the user's home directory, if this file exists, otherwise **/usr/lib/X11/system.hpwmrc**.

doubleClickTime (class **DoubleClickTime**)
This resource is used to set the maximum time (in ms) between the clicks (button presses) that make up a double-click. The default value of this resource is "500" (ms).

iconAutoPlace (class **IconAutoPlace**)
This resource indicates whether icons are automatically placed on the screen by hpwm. If the resource value is "True" then hpwm does automatic icon placement. Users may specify an initial icon position and can move icons, but hpwm will adjust the user-specified position to fit the icon placement scheme (refer to the **IconPlacement** resource). If the resource value is "False" then hpwm does not do automatic icon placement, and the icon placement scheme is ignored. The default value of this resource is "True".

iconDecoration (class **IconDecoration**)
This resource specifies the general icon decoration. The resource value is "label" (only the label part is displayed) or "image" (only the image part is displayed) or "label image" (both the label and image parts are displayed). A value of "activelabel" can also be specified as an enhancement to the "label" value to get a label (not truncated to the width of the icon) when the icon is selected. The default icon decoration is that each icon has a label part and an image part ("label image").

iconImageMaximum (class **IconImageMaximum**)
This resources specifies the maximum size of the icon *image*. The resource value is <width>x<height> (e.g., "64x64"). The default value of this resource is "50x50".

iconImageMinimum (class **IconImageMinimum**)
This resources specifies the minimum size of the icon *image*. The resource value is <width>x<height> (e.g., "32x50"). The default value of this resource is "32x32".

iconPlacement (class **IconPlacement**)
This resource specifies the icon placement scheme to be used. The resource value has the following syntax:

<primary_layout> <secondary_layout>

The layout values are one of the following:

top	Lay the icons out top to bottom.
bottom	Lay the icons out bottom to top.
left	Lay the icons out left to right.
right	Lay the icons out right to left.

A horizontal (vertical) layout value should not be used for both the primary_layout and the secondary_layout (e.g., don't use "top" for the primary_layout and "bottom" for the secondary layout). The primary_layout indicates whether, when an icon placement is done, the icon is placed in a row or a column and the direction of placement. The secondary_layout indicates where to place new rows or columns. For example, "top right" indicates that icons should be placed top to bottom on the screen and that columns should be added from right to left on the screen. The default placement (compatible

with the PM placement policy) is "left bottom" (icons are placed left to right on the screen, with the first row on the bottom of the screen, and new rows added from the bottom of the screen to the top of the screen).

iconPlacementMargin (class **IconPlacementMargin**)
If nonnegative, this resource specifies the distance between the edge of the screen and the icons that are placed along the edge of the screen. Otherwise, this distance is set equal to the space between icons as they are placed on the screen (this space is based on maximizing the number of icons in each row and column). The default value for this resource is -1.

interactivePlacement (class **InteractivePlacement**)
This resource controls the initial placement of new windows on the screen. If it is "True", then the pointer shape changes before a new window is placed on the screen to indicate to the user that a position should be selected for the upper-left hand corner of the window. If "False" then windows will be placed according to the initial window configuration attributes. The default value of this resource is "False."

keyBindings (class **KeyBindings**)
This resource identifies the set of key bindings for window management functions. The named set of key bindings is specified in *hpwm resource description file* file. The default value for this resource is "DefaultKeyBindings".

keyboardFocusPolicy (class **KeyboardFocusPolicy**)
If set to "pointer" the keyboard focus policy is to have the keyboard focus set to the client window that contains the pointer (the pointer could also be in the client window decoration that hpwm adds). If set to "explicit" the policy is to have the keyboard focus set to a client window when the user does a select (Button1 Down) on the client window or any part of the associated hpwm decoration. The default value for this resource is "explicit".

limitResize (class **LimitResize**)
If this resource is "True" the user is not allowed to resize a window to greater than the maximum size (set by the **maximumClientSize** resource or using the WM_NORMAL_HINTS window property). The default value for this resource is "False".

maximumMaximumSize (class **MaximumMaximumSize**)
This resource is used to limit the maximum size of a client window as set by the user or client. The resource value is <width>x<height> (e.g., "1024x1024") where the width and height are in pixels. The default value of this resource is twice the screen width and height.

moveThreshold (class **moveThreshold**)
This resource is used to control the sensitivity of "dragging" operations that are used in moving windows and icons. The value of this resource is the number of pixels that the locator will be moved with a button down before the move operation will be initiated. This is used to prevent window/icon movement when a click or double-click is done and there is unintentional pointer movement with the button down. The default value of this resource is "4" (pixels).

passSelectButton (class **PassSelectButton**)
This resource indicates whether or not the keyboard input focus selection button press (if keyboardFocusPolicy is "explicit") is passed on to the client window or used to do a window management action associated with the window decorations. If the resource value is "False" then the button press will not be used for any operation other than selecting the window to be the keyboard input focus; if the value is "True" the button press will be passed to the client window or used to do a window management operation if appropriate. The default value for this resource is "True".

positionIsFrame (class **PositionIsFrame**)
This resource indicates how client window position information (from the WM_NORMAL_HINTS property and from configuration requests) is to be interpreted.

If the resource value is "True" then the information is interpreted as the position of the hpwm client window frame, if the value is "False" then it is interpreted as being the position of the window. The default value of this resource is "True".

positionOnScreen (class **PositionOnScreen**)
This resource is used to indicate that windows should initially be placed (if possible) so that they are not clipped by the edge of the screen (if the resource value is "True"). If a window is larger then the size of the screen then at least the upper left corner of the window will be on-screen. If the resource value is "False" then windows will be placed in the requested position even if totally off-screen. The default value of this resource is "True".

quitTimeout (class **QuitTimeout**)
This resource specifies the amount of time (in milliseconds) that hpwm will wait for a client to update the WM_COMMAND property after hpwm has sent the WM_SAVE_YOURSELF message. This protocol will only be used for those clients that have a WM_SAVE_YOURSELF atom in the WM_PROTOCOLS client window property. The default value of this resource is "1000" (ms).

resizeBorderWidth (class **ResizeBorderWidth**)
This resource specifies the width (in pixels) of the border around client windows. There will always be some visible border even if this value is set to 0. The default is "10" (pixels).

resizeCursors (class **ResizeCursors**)
This is used to indicate whether the resize cursors are always displayed when the pointer is in the window size border. If "True" the cursors are shown, otherwise the window manager cursor is shown. The default value is "True".

transientDecoration (class **TransientDecoration**)
This controls the amount of decoration that Hpwm puts on transient windows. The gadget specification is exactly the same as for the **clientDecoration** resource (see client class specific resources section below). Transient windows are identified by the WM_TRANSIENT_FOR property which is added by the client to indicate a relatively temporary window. By default, Hpwm will decorate transient windows with a title bar and no other gadgets. The default value for this resource is "title."

Client Class Specific Resources

The syntax for specifying *client class specific resources* for specific classes of clients is:

"Hpwm*<client_class>.<resource_id>"

For example, "Hpwm*HPterm.systemMenu" is used to specify the system menu to be used with hpterm clients.

The syntax for specifying *client class specific resources* for all classes of clients is:

"Hpwm*<resource_id>"

Specific client class specifications take precedence over the specifications for all client classes. For example, "Hpwm*systemMenu" is used to specify the system menu to be used for all classes of clients that don't have a specific system menu specified.

The syntax for specifying resource values for windows that have an unknown class (i.e. the window does not have a WM_CLASS property associated with it) is:

"Hpwm*defaults*<resource_id>"

For example, "Hpwm*defaults*iconImage" is used to specify the icon image to be used for windows that have an unknown class. This is also how a default icon image can be specified for windows that do not have an icon image available from any other source.

The following *client class specific resources* can be specified:

clientDecoration (class **ClientDecoration**)

This resource controls the amount of gadgetry in the client window frame. The resource is specified as a list of gadgets to specify their inclusion in the frame. If a gadget is preceded by a minus sign, then that gadget is excluded from the frame. The *sign* of the first item in the list determines the initial amount of gadgetry. If the sign of the first item is minus, then hpwm assumes all gadgets present and starts subtracting from that set. If the sign of the first item is plus (or not specified), then hpwm starts with no gadgets and builds up a list from the resource.

Name	Description
all	Include all gadgets
maximize	Maximize box (includes title bar)
minimize	Minimize box (includes title bar)
none	No gadgets
resize	Resize border
system	System menu box (includes title bar)
title	Title bar (only)

The default value for this resource is "all."

iconImage (class **IconImage**)

This resource can be used to specify an icon image for a particular class of clients (i.e. "Hpwm*<client_class>.iconImage") or an icon image to be used for all classes of clients that don't have a specifically specified icon image (i.e. "Hpwm*iconImage"). The resource value is a pathname for a bitmap file. If specified this resource overrides any client specified icon image.

The default value is to display the client supplied icon image if it is defined; if a client icon image is not available then the icon image specified by "Hpwm*defaults*iconImage" is used if it is specified; if an icon image is not supplied by the user or the client then a built-in window manager icon image is used.

iconImageBackground (class **Background**)

This resource specifies the background color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon background color (i.e. specified by "Hpwm*background or Hpwm*icon*background).

iconImageBottomShadowColor (class **Foreground**)

This resource specifies the bottom shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow color (i.e. specified by Hpwm*bottomShadowColor or Hpwm*icon*bottomShadowColor).

iconImageBottomShadowTile (class **BottomShadowTile**)

This resource specifies the bottom shadow tile of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow tile (i.e. specified by Hpwm*bottomShadowTile or Hpwm*icon*bottomShadowTile).

iconImageForeground (class **Foreground**)

This resource specifies the foreground color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon foreground color (i.e. specified by "Hpwm*foreground or Hpwm*icon*foreground).

iconImageTopShadowColor (class **Background**)

This resource specifies the top shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow color (i.e. specified by Hpwm*topShadowColor or Hpwm*icon*topShadowColor).

iconImageTopShadowTile (class **TopShadowTile**)

This resource specifies the top shadow tile of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow tile (i.e.

specified by Hpwm*topShadowTile or Hpwm*icon*topShadowTile).

makeIconColors (class **MakeColors**)
If the value of this resource is "all" then hpwm will use the value of the **iconImageBackground** resource to make values for the **iconImageBottomShadowColor, iconImageForeground** and **iconImageTopShadowColor** resources that provide a 3-D appearance. In this case the **iconImageTopShadowTile** and **iconImageBottomShadowTile** are always set to "foreground". If the makeColors resource value is "shadow" the top and bottom shadow colors will be made but the foreground color will not be made. If the makeColors resource value is "none" then no colors will be automatically made. The default value for this resource is "shadow".

makeMatteColors (class **MakeColors**)
If the value of this resource is "all" then hpwm will use the value of the **matteBackground** resource to make values for the **matteBottomShadowColor, matteForeground** and **matteTopShadowColor** resources that provide a 3-D appearance. In this case the **matteTopShadowTile** and **matteBottomShadowTile** are always set to "foreground". If the makeColors resource value is "shadow" the top and bottom shadow colors will be made but the foreground color will not be made. If the makeColors resource value is "none" then no colors will be automatically made. The default value for this resource is "shadow".

matteBackground (class **Background**)
This resource specifies the background color of the matte, when **matteWidth** is positive. The default value of this resource is the client background color (i.e. specified by "Hpwm*background or Hpwm*client.background).

matteBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the matte, when **matteWidth** is positive. The default value of this resource is the client bottom shadow color (i.e. specified by "Hpwm*bottomShadowColor or Hpwm*client.bottomShadowColor).

matteBottomShadowTile (class **BottomShadowTile**)
This resource specifies the bottom shadow tile of the matte, when **matteWidth** is positive. The default value of this resource is the client bottom shadow tile (i.e. specified by "Hpwm*bottomShadowTile or Hpwm*client.bottomShadowTile).

matteForeground (class **Foreground**)
This resource specifies the foreground color of the matte, when **matteWidth** is positive. The default value of this resource is the client foreground color (i.e. specified by "Hpwm*foreground or Hpwm*client.foreground).

matteTopShadowColor (class **Background**)
This resource specifies the top shadow color of the matte, when **matteWidth** is positive. The default value of this resource is the client top shadow color (i.e. specified by "Hpwm*topShadowColor or Hpwm*client.topShadowColor).

matteTopShadowTile (class **TopShadowTile**)
This resource specifies the top shadow tile of the matte, when **matteWidth** is positive. The default value of this resource is the client top shadow tile (i.e. specified by "Hpwm*topShadowTile or Hpwm*client.topShadowTile).

matteWidth (class **MatteWidth**)
This resource specifies the width of the optional *matte* that can be specified to frame the client area of the window. The matte sits just inside the other client window frame decoration. This is useful to help old applications fit into a suite of new applications that use the HP widget set. The matte is given a 3-D effect just like the HP widgets. The default value is 0, which effectively disables the matte.

maximumClientSize (class **MaximumClientSize**)
This is a size specification that indicates the client size to be used when an application is maximized. The resource value is specified as "<width>x<height>". If this resource is not specified then the maximum size from the WM_NORMAL_HINTS property is used if set. Otherwise the default value is the size where the client window with window

management borders fills the screen.

systemMenu (class **SystemMenu**)
This resource indicates the name of the menu pane that is posted when the system menu is popped up (usually by pressing button 1 on the system box gadget on the client window frame). Menu panes are specified in the *hpwm resource description file* file. System Menus can be customized on a client class basis by specifying resources of the form Hpwm*<client class>.systemMenu (See **Hpwm Resource Description File Syntax**). The default value of this resource is "DefaultSystemMenu".

RESOURCE DESCRIPTION FILE

The *hpwm resource description file* is a supplementary resource file that contains resource descriptions that are referred to by entries in the defaults files (.Xdefaults, app-defaults/Hpwm). It contains descriptions of resources that are to be used by hpwm, and that cannot be easily encoded in the defaults files (a bitmap file is an analogous type of resource description file). A particular *hpwm resource description file* can be selected using the **configFile** resource.

The following types of resources can be described in the *hpwm resource description file*:

Buttons	Window manager functions can be bound (associated) with button press events.
Keys	Window manager functions can be bound (associated) with key press events.
Menu	These menu panes can be used for the system menu and other menus posted with key and button bindings.

Hpwm Resource Description File Syntax

The *hpwm resource description file* is a standard text file that contains items of information separated by blanks, tabs and new lines characters. Blank lines are ignored. Items or characters can be quoted to avoid special interpretation (e.g., the comment character can be quoted to prevent it from being interpreted as the comment character). A quoted item can be contained in double quotes ("). Single characters can be quoted by preceding them by the back-slash character (\). All text from an unquoted # to the end of the line is regarded as a comment and is not interpreted as part of a resource description. Window manager functions can be accessed with button and key bindings, and with window manager menus. Functions are indicated as part of the specifications for botton and key binding sets, and menu panes. The function specification has the following syntax:

```
function =          function_name [function_args]
function_name =     <window manager function>
function_args =     {quoted | unquoted_item}
```

The following functions are supported. If a function is specified that isn't one of the supported functions then it is interpreted by hpwm as *f.nop*.

f.beep	This function beeps.
f.circle_down	This function lowers the highest mapped client window that partially or completely obscures another client window to the bottom of the window stack (where it obscures no other window).
f.circle_up	This function raises the lowest mapped client window that is partially or completely obscured by another client window to the top of the window stack (where it is obscured by no other window).
f.exec or !	This function causes *function-args* to be executed (using */bin/sh*).
f.focus_color	This function sets the colormap focus to a client window. If this function is done in a root context then the default colormap (setup by the *X Window System* for the screen where hpwm is running) will be installed and there will be no specific client window colormap focus. This function is treated as *f.nop* if *colormapFocusPolicy* is not "explicit".

f.focus_key This function sets the keyboard input focus to a client window or icon. This function is treated as *f.nop* if *keyboardFocusPolicy* is not "explicit" or the function is executed in a root context.

f.kill This function causes a client's X connection to be terminated (usually resulting in termination of the client).

f.lower This function lowers a client window to the bottom of the window stack (where it obscures no other window).

f.maximize This function causes a client window to be displayed with its maximum size.

f.menu If this function appears in a menu pane entry, it associates the cascading (pull-right) menu identified by function_args with the menu pane entry. If this function appears in a key or button binding, it posts the menu identified by function_args.

f.minimize This function causes a client window to be minimized / iconized.

f.move This function allows a client window to be interactively moved.

f.next_cmap This function installs the next colormap in the list of colormaps for the window with the colormap focus.

f.next_key This function sets the keyboard input focus to the next window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen).

f.nop This function does nothing.

f.normalize This function causes a client window to be displayed with its normal size.

f.post_smenu This function is used to post the system menu.

f.prev_key This function sets the keyboard input focus to the previous window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen).

f.quit_hpwm This function terminates hpwm (but NOT the X window system).

f.raise This function raises a client window to the top of the window stack (where it is obscured by no other window).

f.raise_lower This function raises an obscured client window to the top of the window stack (where it is obscured by no other window) and lowers a client window that is on top of the window stack to the bottom of the stack (where it obscures no other window).

f.refresh This function causes all windows to be redrawn.

f.refresh_win This function causes a client window to be redrawn.

f.resize This function allows a client window to be interactively resized.

f.restart This function causes hpwm to be restarted (effectively terminated and re-exec'ed).

f.separator This function causes a menu separator to be put in the menu pane entry (the label is ignored).

f.title This function inserts a title in the menu pane if it is the first entry in the menu pane description. By default a menu pane has no title.

Each function may be constrained as to which resource types can specify the function (e.g., menu pane), and also what context the function can be used in (e.g., the function is done to the selected client window). Function contexts are:

root No client window or icon has been selected as an object for the function.

window A client window has been selected as an object for the function. This includes the window's title bar and frame. Some functions are applied only when the window is in its normalized state (e.g., f.maximize) or its maximized state (e.g.,

f.normalize).

icon An icon has been selected as an object for the function.

frame A client window frame has been selected as an object for the function. This includes the window frame's title bar.

title A client window title bar has been selected as an object for the function.

If a function is specified in a type of resource where it is not supported or is invoked in a context that does not apply then the function is treated as *f.nop*. The following table indicates the resource types and function contexts in which window manager functions apply.

Function	Contexts	Resources
f.beep	root,icon,window	button,key,menu
f.circle_down	root,icon,window	button,key,menu
f.circle_up	root,icon,window	button,key,menu
f.exec	root,icon,window	button,key,menu
f.focus_color	root,icon,window	button,key,menu
f.focus_key	icon,window	button,key,menu
f.kill	icon,window	menu
f.lower	icon,window	button,key,menu
f.maximize	icon,window(normal)	button,key,menu
f.menu	root,icon,window	button,key,menu
f.minimize	window	button,key,menu
f.move	icon,window	button,key,menu
f.next_cmap	root,icon,window	button,key,menu
f.next_key	root,icon,window	button,key,menu
f.nop	root,icon,window	button,key,menu
f.normalize	icon,window(maximized)	button,key,menu
f.post_smenu	root,icon,window	button,key
f.prev_key	root,icon,window	button,key,menu
f.quit_hpwm	root	menu
f.raise	icon,window	button,key,menu
f.raise_lower	icon,window	button,key,menu
f.refresh	root,icon,window	button,key,menu
f.refresh_win	window	button,key,menu
f.resize	window	button,key,menu
f.restart	root	menu
f.separator	root,icon,window	menu
f.title	root,icon,window	menu

Button Bindings

A window manager function can be done with a pointer button press or release when the pointer is over a client window, an icon or the root window. The context for indicating where the button action applies is also the context for invoking the window manager function when the button press is done.

The button binding syntax is:

```
Buttons bindings_set_name
{
  button   context   function
  button   context   function
                .
                .
  button   context   function
}
```

The *button* specification is done using the syntax supported by the *X Toolkit's* translation manager, with three exceptions: only a single event may be specified, the event must be a *ButtonPress* or a *ButtonRelease*, and all modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the button event occurs). For example, button 1 down with the [Shift] key pressed is specified by: Shift<Btn1Down> and button 1 up with the [Extend char] key pressed is specified by Meta<Btn1Up>. Button release specifications are interpreted by hpwm as a "click" (i.e. a button press followed by a button release with less than the move threshold amount of motion in between).

The syntax for the *context* specification is:

```
context =   object["|"context]
object =    root | icon | window | title | frame
```

The context specification indicates where the pointer must be for the button binding to be effective. The *frame* context is for the client window frame and the *title* context is for the title area of the client window frame. For example, a context of *window* indicates that the pointer must be over a client window or window frame or window title for the button binding to be effective. For button bindings the *frame* and *title* contexts are equivalent to the *window* context.

If a *f.nop* function is specified for a button binding the button binding will not be done.

Key Bindings

A window manager function can be done when a particular key is pressed. The context in which the key binding applies is indicated in the key binding specification. The valid contexts are the same as those that apply to button bindings.

The key binding syntax is:

```
Keys bindings_set_name
{
  key   context   function
  key   context   function
          .
          .
  key   context   function
}
```

The *key* specification is done using the syntax supported by the *X Toolkit's* translation manager, with three exceptions: only a single event may be specified, the event must be a key event, and all modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the key event occurs). For example, a shifted [C] key press is specified by Shift<Key>c.

If a *f.nop* function is specified for a key binding the key binding will not be done. If a *f.post_smenu* or *f.menu* function is bound to a key, hpwm will automatically use the same key for removing the menu from the screen after it has been popped up.

The *context* specification syntax is the same as for button bindings. For key bindings the *frame* and *title* contexts are equivalent to the *window* context.

Menu Panes

The context for window manager functions that are done from the system menu is *icon* or *window* depending on where the system menu was popped up. For other menus the context depends on the location of the pointer (for menus posted by button bindings) or the location of the keyboard input focus (for menus posted by key bindings).

The menu pane specification syntax is:

```
Menu menu_pane_name
{
  label   function
  label   function
          .
          .
  label   function
}
```

Each line in the *Menu* specification identifies the label for a menu item and the function to be done if the menu item is selected. The *label* may be a string or a bitmap file. The label specification has the following syntax:

```
label =         text | bitmap_file
bitmap_file =   "@"<file_name>
text =          quoted_item | unquoted_item
```

The sting encoding for labels must be compatible with the menu font that is used. Labels are greyed out for menu items that do the *f.nop* function or an invalid function or a function that doesn't apply in the current context.

WINDOW MANAGER COMMANDS

The user interactively commands the window manager with the keyboard and pointer. This interface can be configured by the user by changing entries in the resource files for hpwm (see above).

Pointer Commands

The pointer (usually a mouse) is the device that controls a cursor that ranges over the entire screen. Hpwm uses the pointer in the following ways:

Button 1 Click
Select an object/action. The default actions performed by doing a button 1 click on a window manager object are:

Object	Action
window frame	top window
icon	top icon
window frame and window	explicit keyboard focus selection
icon	explicit keyboard focus selection
frame minimize gadget	minimize the window
frame maximize gadget	maximize the window

Button 1 Double-click
The default action performed by doing a button 1 double-click on an icon is to normalize the associated window.

Button 1 Drag
Select and perform a move/resize action or pop up a menu and select a menu item. The default actions performed by doing a button 1 drag on a window manager object are:

Object	Action
window frame title	move the window

window resize border	resize the window
icon	move the icon
frame system gadget	popup the system menu, select an item

The **buttonBindings** resource can be used to associate window management functions with button presses (although default, unmodified Button 1 function bindings supercede unmodified Button 1 *buttonBindings* specifications). This is useful for users who do not want screen space taken up by window decoration. This mechanism allows them to run without decoration, but still have a way of manipulating windows. The default button bindings are:

Button	Function	Context	Description
Button1	f.menu	root	post the default root window menu
Button1 Click	f.raise	frame	raise window to top of stack
Meta-Button1 Drag	f.move	window	move a window
Meta-Button3 Click	f.minimize	window	minimize a window

Keyboard Commands

Keyboard bindings for window management functions can be defined to allow window management to be done without using a pointer. The resource **keyBindings** is used to associate key presses with window management functions. The default key bindings for window manipulation are:

Key	Function	Description
Shift-Escape	f.post_smenu	pop up the system menu
Meta-Tab	f.next_key	go to next window in stack
Return	<built-in>	accept menu selection
Up-Arrow	<built-in>	move to previous item in menu
Down-Arrow	<built-in>	move to next item in menu

Key bindings that are used to pop up the system menu are automatically used by hpwm as key bindings for removing the menu when it is popped up.

The f.next_key function is only valid when **keyboardFocusPolicy** is **explicit**. No such function is provided when **keyboardFocusPolicy** is **pointer**.

Interactive Window Placement

The user has the option of interactively positioning and sizing new windows before they appear on the display. When **interactivePlacement** resource is set to "True", the pointer shape will change before the new window is mapped. A window will appear in the center of the screen that provides position and size feedback.

The user sets the initial position by moving the pointer to the desired location and clicking the Select button. The user may set the initial size of the window by depressing the Select button to fix the location of the upper left-hand corner of the window and then dragging the pointer to the desired position of the lower right-hand corner of the window. The window size will be fixed once the user releases the Select button. The **moveThreshold** resource will be used to distinguish between an accidental movement of the pointer while clicking versus a deliberate size setting action.

You can also do interactive placement strictly from the keyboard. The arrow keys move the pointer in the expected direction. Arrow keys while the [CTRL] key is held down to moves the pointer in large increments. The space bar stops moving the window and gets hpwm ready to change its size. The [Return] key completes interactive placement at any time.

The feedback window provides position and size information as the pointer is moved. The size feedback will be in multiples of width_inc and height_inc if those values are defined in the WM_NORMAL_HINTS for the window. In addition, a wire frame of the window is drawn to give

the user graphic feedback of the window size.

If the window position information is specified when the client is invoked, then interactive placement will not take place. The window will be placed at the position specified. The window will be of the default size unless the user also specifies this along with the position. If the user only specifies the size of the window, hpwm will use that size as the default during interactive placement.

Hpwm will not do interactive placement on windows that have the WM_TRANSIENT_FOR property set. These are assumed to exist for short duration interactions (dialog boxes) with which interactive placement would interfere.

ENVIRONMENT

Hpwm uses the environment variable **$HOME** specifying the user's home directory.

FILES

/usr/lib/X11/system.hpwmrc
/usr/lib/X11/app-defaults/Hpwm
$HOME/.Xdefaults

COPYRIGHT

ORIGIN

Hewlett-Packard Company

SEE ALSO

X(1)

Series 300 and 800 Only

NAME

mkfontdir - create fonts.dir file from directory of font files.

SYNOPSIS

mkfontdir [directory-names]

DESCRIPTION

For each directory argument, *mkfontdir* reads all of the font files in the directory searching for properties named "FONT", or (failing that) the name of the file stripped of its suffix. These are used as font names, which are written out to the file "fonts.dir" in the directory along with the name of the font file. Without a "fonts.dir" file, the server will not be able to access the font files in the directory.

The kinds of font files read by mkfontdir depends on configuration parameters, but typically include SNF (suffix ".snf"), compressed SNF (suffix ".snf.Z", or ".scf"), BDF (suffix ".bdf"), and compressed BDF (suffix ".bdf.Z"). If a font exists in multiple formats, the most efficient format will be used.

FONT NAME ALIASES

The file "fonts.alias" which can be put in any directory of the font-path is used to map new names to existing fonts, and should be edited by hand. The format is straight forward enough, two white-space separated columns, the first containing aliases and the second containing font-name patterns.

When a font alias is used, the name it references is searched for in the normal manner, looking through each font directory in turn. This means that the aliases need not mention fonts in the same directory as the alias file.

To embed white-space in either name, simply enclose them in double-quote marks, to embed double-quote marks (or any other character), preceed them with back-slash:

```
"magic-alias with spaces" "\"font\name\" with quotes"
regular-alias                    fixed
```

If the string "FILE_NAMES_ALIASES" stands alone on a line, each file-name in the directory (stripped of it's .snf suffix) will be used as an alias for that font.

USAGE

Xserver(1) looks for both "fonts.dir" and "fonts.alias" in each directory in the font path each time it is set (see xset(1)).

SEE ALSO

X(1), Xserver(1), xset(1)

NAME

mwm - The OSF/Motif Window Manager.

SYNOPSIS

mwm [options]

DESCRIPTION

The OSF/Motif Window Manager (*mwm*) is an X11 client that provides window management functionality and some session management functionality. It provides functions that facilitate control (by the user and the programmer) of elements of window states such as placement, size, icon/normal display, input focus ownership, etc. It also provides session management functions such as stopping a client.

OPTIONS

-display *display*
This option specifies the display to use; see *X(1)*.

-xrm *resourcestring*
This option specifies a resource string to use.

APPEARANCE

The following sections describe the basic default behaviors of windows, icons, the icon box, input focus, and window stacking. The appearance and behavior of the window manager can be altered by changing the configuration of specific resources. Resources are defined under the heading "X DEFAULTS."

WINDOWS

Default mwm window frames have distinct components with associated functions:

Title Area	In addition to displaying the client's title, the title area is used to move the window. To move the window, place the pointer over the title area, press button 1 and drag the window to a new location. A wire frame is moved during the drag to indicate the new location. When the button is released, the window is moved to the new location.
Title Bar	The title bar includes the title area, the minimize button, the maximize button and the window menu button.
Minimize Button	To turn the window back into its icon, do a button 1 click on the minimize button (the frame box with a *small* square in it).
Maximize Button	To make the window fill the screen (or enlarge to the largest size allowed by the configuration files), do a button 1 click on the maximize button (the frame box with a *large* square in it).
Window Menu Button	The window menu button is the frame box with a horizontal bar in it. To pop up the window menu, press button 1. While pressing, drag the pointer on the menu to your selection, then release the button when your selection is highlighted. Alternately, you can click button 1 to pop up the menu and keep it posted; then position the pointer and select.

Default Window Menu		
Selection	**Accelerator**	**Description**
Restore	Alt+F5	Inactive (not an option for windows).
Move	Alt+F7	Allows the window to be moved with keys or mouse.
	Shift + Move Grow	Apollo keyboard.
Size	Alt+F8	Allows the window to be resized.
	Move Grow	Apollo keyboard.
Minimize	Alt+F9	Turns the window into an icon.
	Shift + Pop	Apollo keyboard.
Maximize	Alt+F10	Makes the window fill the screen.
	Ctrl + Move Grow	Apollo keyboard.
Lower	Alt+F3	Moves window to bottom of window stack.
Close	Alt+F4	Removes client from MWM management.

Resize Border Handles To change the size of a window, move the pointer over a resize border handle (the cursor will change), press button 1, and drag the window to a new size. When the button is released, the window is resized. While dragging is being done, a rubber-band outline is displayed to indicate the new window size.

Matte An optional matte decoration can be added between the client area and the window frame. A matte is not actually part of the window frame. There is no functionality associated with a matte.

ICONS

Icons are small graphic representations of windows. A window can be minimized (iconified) using the minimize button on the window frame. Icons provide a way to reduce clutter on the screen.

Pressing mouse button 1 when the pointer is over an icon will cause the icon's window menu to pop up. Releasing the button (press + release without moving mouse = click) will cause the menu to stay posted. The menu contains the following selections:

Series 300 and 800 Only

Icon Window Menu		
Selection	**Accelerator**	**Description**
Restore	Alt+F5	Opens the associated window.
	Shift+ Pop	Apollo keyboard.
Move	Alt+F7	Allows the icon to be moved with keys.
	Shift + Move Grow	Apollo keyboard.
Size	Alt+F8	Inactive (not an option for icons).
Minimize	Alt+F9	Inactive (not an option for icons).
Maximize	Alt+F10	Opens the associated window and makes it fill the screen.
	Ctrl + Pop	Apollo keyboard.
Lower	Alt+F3	Moves icon to bottom of icon stack.
Close	Alt+F4	Removes client from MWM management.

Double-clicking button 1 on an icon normalizes the icon into its associated window. Double-clicking button 1 on the icon box's icon opens the icon box and allow access to the contained icons. (In general, double-clicking a mouse button offers a quick way to have a function performed. Another example is double-clicking button 1 with the pointer on the window menu button. This closes the window.)

ICON BOX

When icons begin to clutter the screen, they can be packed into an "icon box." (To use an icon box, MWM must be started with the icon box configuration already set.) The icon box is a window manager window that holds client icons. Icons in the icon box can be manipulated with the mouse. The following table summarizes the behavior of this interface. Button actions apply whenever the pointer is on any part of the icon.

Button Action	Description
Button 1 click	Selects the icon.
Button 1 double click	Normalizes (opens) the associated window.
Button 1 double click	Raises an already *open* window to the top of the stack.
Button 1 drag	Moves the icon.

The window menu of the icon box differs from the window menu of a client window: The "Close" selection is replaced with the "PackIcons Alt+F12" selection. When selected, PackIcons packs the icons in the box to achieve neat rows with no empty slots.

INPUT FOCUS

Mwm supports (by default) a keyboard input focus policy of explicit selection. This means when a window is selected to get keyboard input, it continues to get keyboard input until the window is withdrawn from window management, another window is explicitly selected to get keyboard input, or the window is iconified. There are numerous resources that control the input focus. The client window with the keyboard input focus has the active window appearance with a visually distinctive window frame.

The following tables summarize the keyboard input focus selection behavior:

Series 300 and 800 Only

Button Action	Object	Function Description
Button 1 press	Window / window frame	Keyboard focus selection
Button 1 press	Icon	Keyboard focus selection

Key Action	Function Description
[Alt][Tab]	Move input focus to next window in window stack.
[Next Wndw]	Apollo keyboard.
[Alt][Shift][Tab]	Move input focus to previous window in window stack.
[Shift][Next Wndw]	Apollo keyboard.

WINDOW STACKING

The stacking order of windows may be changed as a result of setting the keyboard input focus, iconifying a window, or by doing a window manager window stacking function.

When a window is iconified, the window's icon is placed on the bottom of the stack.

The following table summarizes the default window stacking behavior of the window manager:

Key Action	Function Description
[Alt][ESC]	Put bottom window on top of stack.
[Ctrl][Pop]	Apollo keyboard.
[Alt][Shift][ESC]	Put top window on bottom of stack.
[Alt][Pop]	Apollo keyboard.
[Pop]	Apollo keyboard. Put bottom window on top of stack; put top window on bottom of stack.

A window can also be raised to the top when it gets the keyboard input focus (e.g., by doing a button 1 press on the window or by using [Alt][Tab]) if this auto-raise feature is enabled with the **focusAutoRaise** resource.

X DEFAULTS

Mwm is configured from its resource database. This database is built from the following sources. They are listed in order of precedence, low to high:

app-defaults/Mwm
RESOURCE_MANAGER root window property or $HOME/.Xdefaults
XENVIRONMENT variable or $HOME/.Xdefaults-host
mwm command line options

Entries in the resource database may refer to other resource files for specific types of resources. These include files that contain bitmaps, fonts, and mwm specific resources such as menus and behavior specifications (i.e., button and key bindings).

Mwm is the resource class name of mwm and **mwm** is the resource name used by mwm to look up resources. In the following discussion of resource specification "Mwm" and "mwm" can be used interchangeably.

Mwm uses the following types of resources:

Component Appearance Resources:

These resources specify appearance attributes of window manager user interface components. They can be applied to the appearance of window manager menus, feedback windows (e.g., the window reconfiguration feedback window), client window frames, and icons.

Specific Appearance and Behavior Resources:

These resources specify mwm appearance and behavior (e.g., window management policies). They are not set separately for different mwm user interface components.

Client Specific Resources:

These mwm resources can be set for a particular client window or class of client windows. They specify client-specific icon and client window frame appearance and behavior.

Resource identifiers can be either a resource name (e.g., foreground) or a resource class (e.g., Foreground). If the value of a resource is a filename and if the filename is prefixed by "~/", then it is relative to the path contained in the *$HOME* environment variable (generally the user's home directory). This is the only environment variable mwm uses directly ($XENVIRONMENT is used by the resource manager).

COMPONENT APPEARANCE RESOURCES

The syntax for specifying *component appearance resources* that apply to window manager icons, menus, and client window frames is

Mwm*resource_id*

For example, **Mwm*foreground** is used to specify the foreground color for mwm menus, icons, and client window frames.

The syntax for specifying *component appearance resources* that apply to a particular mwm component is

Mwm*[menu|icon|client|feedback]*resource_id*

If *menu* is specified, the resource is applied only to mwm menus; if *icon* is specified, the resource is applied to icons; and if *client* is specified, the resource is applied to client window frames. For example, **Mwm*icon*foreground** is used to specify the foreground color for mwm icons, **Mwm*menu*foreground** specifies the foreground color for mwm menus, and **Mwm*client*foreground** is used to specify the foreground color for mwm client window frames.

The appearance of the title area of a client window frame (including window management buttons) can be separately configured. The syntax for configuring the title area of a client window frame is:

Mwm*client*title*resource_id*

For example, **Mwm*client*title*foreground** specifies the foreground color for the title area. Defaults for title area resources are based on the values of the corresponding client window frame resources.

The appearance of menus can be configured based on the name of the menu. The syntax for specifying menu appearance by name is:

Mwm*menu*menu_name**resource_id*

For example, **Mwm*menu*my_menu*foreground** specifies the foreground color for the menu named **my_menu**.

The following *component appearance resources* that apply to all window manager parts can be specified:

Component Appearance Resources - All Window Manager Parts			
Name	**Class**	**Value Type**	**Default**
background	Background	color	varies*
backgroundPixmap	BackgroundPixmap	string**	varies*
bottomShadowColor	Foreground	color	varies*
bottomShadowPixmap	BottomShadowPixmap	string**	varies*
fontList	FontList	string***	"fixed"
foreground	Foreground	color	varies*
saveUnder	SaveUnder	T/F	F
topShadowColor	Background	color	varies*
topShadowPixmap	TopShadowPixmap	string**	varies*

*The default is chosen based on the visual type of the screen. **Pixmap image name. See XmInstallImage(3X).
***X11 R3 Font description.

background (class **Background**)
This resource specifies the background color. Any legal X color may be specified. The default value is chosen based on the visual type of the screen.

backgroundPixmap (class **BackgroundPixmap**)
This resource specifies the background Pixmap of the mwm decoration when the window is inactive (does not have the keyboard focus). The default value is chosen based on the visual type of the screen.

bottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color. This color is used for the lower and right bevels of the window manager decoration. Any legal X color may be specified. The default value is chosen based on the visual type of the screen.

bottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow Pixmap. This Pixmap is used for the lower and right bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

fontList (class **Font**)
This resource specifies the font used in the window manager decoration. The character encoding of the font should match the character encoding of the strings that are used. The default is "fixed."

foreground (class **Foreground**)
This resource specifies the foreground color. The default is chosen based on the visual type of the screen.

saveUnder (class **SaveUnder**)
This is used to indicate whether "save unders" are used for mwm components. For this to have any effect, save unders must be implemented by the X server. If save unders are implemented, the X server will save the contents of windows obscured by windows that have the save under attribute set. If the saveUnder resource is True, mwm will set the save under attribute on the window manager frame of any client that has it set. If saveUnder is False, save unders will not be used on any window manager frames. The default value is False.

topShadowColor (class **Background**)
This resource specifies the top shadow color. This color is used for the upper and left bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

topShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow Pixmap. This Pixmap is used for the upper and left bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

The following *component appearance resources* that apply to frame and icons can be specified:

Frame and Icon Components			
Name	Class	Value Type	Default
activeBackground	Background	color	varies*
activeBackgroundPixmap	BackgroundPixmap	string**	varies*
activeBottomShadowColor	Foreground	color	varies*
activeBottomShadowPixmap	BottomShadowPixmap	string**	varies*
activeForeground	Foreground	color	varies*
activeTopShadowColor	Background	color	varies*
activeTopShadowPixmap	TopShadowPixmap	string**	varies*

*The default is chosen based on the visual type of the screen. **See XmInstallImage(3X).

activeBackground (class **Background**)
This resource specifies the background color of the mwm decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBackgroundPixmap (class **ActiveBackgroundPixmap**)
This resource specifies the background Pixmap of the mwm decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the mwm decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow Pixmap of the mwm decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeForeground (class **Foreground**)
This resource specifies the foreground color of the mwm decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeTopShadowColor (class **Background**)
This resource specifies the top shadow color of the mwm decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeTopShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow Pixmap of the mwm decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

SPECIFIC APPEARANCE AND BEHAVIOR RESOURCES

The syntax for specifying *specific appearance and behavior resources* is

Mwm*_resource_id_

For example, **Mwm*keyboardFocusPolicy** specifies the window manager policy for setting the keyboard focus to a particular client window.

The following *specific appearance and behavior resources* can be specified:

Series 300 and 800 Only

Specific Appearance and Behavior Resources			
Name	**Class**	**Value Type**	**Default**
autoKeyFocus	AutoKeyFocus	T/F	T
autoRaiseDelay	AutoRaiseDelay	millisec	500
bitmapDirectory	BitmapDirectory	directory	/usr/include/X11/bitmaps
buttonBindings	ButtonBindings	string	NULL
cleanText	CleanText	T/F	T
clientAutoPlace	ClientAutoPlace	T/F	T
colormapFocusPolicy	ColormapFocusPolicy	string	keyboard
configFile	ConfigFile	file	.mwmrc
deiconifyKeyFocus	DeiconifyKeyFocus	T/F	T
doubleClickTime	DoubleClickTime	millisec.	500
enforceKeyFocus	EnforceKeyFocus	T/F	T
fadeNormalIcon	FadeNormalIcon	T/F	F
frameBorderWidth	FrameBorderWidth	pixels	5
iconAutoPlace	IconAutoPlace	T/F	T
iconBoxGeometry	IconBoxGeometry	string	6x1+0-0
iconBoxName	IconBoxName	string	iconbox
iconBoxTitle	IconBoxTitle	string	Icons
iconClick	IconClick	T/F	T
iconDecoration	IconDecoration	string	varies
iconImageMaximum	IconImageMaximum	wxh	50x50
iconImageMinimum	IconImageMinimum	wxh	32x32
iconPlacement	IconPlacement	string	left bottom
iconPlacementMargin	IconPlacementMargin	pixels	varies
interactivePlacement	InteractivePlacement	T/F	F
keyBindings	KeyBindings	string	Motif
keyboardFocusPolicy	KeyboardFocusPolicy	string	explicit
limitResize	LimitResize	T/F	T
lowerOnIconify	LowerOnIconify	T/F	T
maximumMaximumSize	MaximumMaximumSize	wxh (pixels)	2X screen w&h
moveThreshold	MoveThreshold	pixels	4
passButtons	PassButtons	T/F	F
passSelectButton	PassSelectButton	T/F	T
positionIsFrame	PositionIsFrame	T/F	T
positionOnScreen	PositionOnScreen	T/F	T
quitTimeout	QuitTimeout	millisec.	1000
resizeBorderWidth	ResizeBorderWidth	pixels	10
resizeCursors	ResizeCursors	T/F	T
showFeedback	ShowFeedback	string	all
startupKeyFocus	StartupKeyFocus	T/F	T
transientDecoration	TransientDecoration	string	system title
transientFunctions	TransientFunctions	string	-minimize -maximize
useIconBox	UseIconBox	T/F	F
wMenuButtonClick	WMenuButtonClick	T/F	T
wMenuButtonClick2	WMenuButtonClick2	T/F	T

autoKeyFocus (class **AutoKeyFocus**)

This resource is only available when the keyboard input focus policy is explicit. If autoKeyFocus is given a value of True, then when a window with the keyboard input focus is withdrawn from window management or is iconified, the focus is set to the previous window that had the focus. If the value given is False, there is no automatic

setting of the keyboard input focus. The default value is True.

autoRaiseDelay (class **AutoRaiseDelay**)
This resource is only available when the focusAutoRaise resource is True and the keyboard focus policy is pointer. The autoRaiseDelay resource specifies the amount of time (in milliseconds) that mwm will wait before raising a window after it gets the keyboard focus. The default value of this resource is 500 (ms).

bitmapDirectory (class **BitmapDirectory**)
This resource identifies a directory to be searched for bitmaps referenced by mwm resources. This directory is searched if a bitmap is specified without an absolute pathname. The default value for this resource is "/usr/include/X11/bitmaps".

buttonBindings (class **ButtonBindings**)
This resource identifies the set of button bindings for window management functions. The named set of button bindings is specified in the *mwm resource description file*. These button bindings are *merged* with the built-in default bindings. The default value for this resource is NULL (i.e., no button bindings are added to the built-in button bindings).

cleanText (class**CleanText**)
This resource controls the display of window manager text in the client title and feedback windows. If the default value of True is used, the text is drawn with a clear (no stipple) background. This makes text easier to read on monochrome systems where a backgroundPixmap is specified. Only the stippling in the area immediately around the text is cleared. If False, the text is drawn directly on top of the existing background.

clientAutoPlace (class **ClientAutoPlace**)
This resource determines the position of a window when the window has not been given a user specified position. With a value of True, windows are positioned with the top left corners of the frames offset horizontally and vertically. A value of False causes the currently configured position of the window to be used. In either case, MWM will attempt to place the windows totally on-screen. The default value is True.

colormapFocusPolicy (class **ColormapFocusPolicy**)
This resource indicates the colormap focus policy that is to be used. If the resource value is explicit then a colormap selection action is done on a client window to set the colormap focus to that window. If the value is pointer then the client window containing the pointer has the colormap focus. If the value is keyboard then the client window that has the keyboard input focus will have the colormap focus. The default value for this resource is keyboard.

configFile (class **ConfigFile**)
The resource value is the pathname for an *mwm resource description file*. The default is **.mwmrc** in the user's home directory (based on the $HOME environment variable) if this file exists, otherwise **/usr/lib/X11/system.mwmrc**.

deiconifyKeyFocus (class **DeiconifyKeyFocus**)
This resource only applies when the keyboard input focus policy is explicit. If a value of True is used, a window will receive the keyboard input focus when it is normalized (deiconified). True is the default value.

doubleClickTime (class **DoubleClickTime**)
This resource is used to set the maximum time (in ms) between the clicks (button presses) that make up a double-click. The default value of this resource is 500 (ms).

enforceKeyFocus (class **EnforceKeyFocus**)
If this resource is given a value of True, then the keyboard input focus is always explicitly set to selected windows even if there is an indication that they are "globally active" input windows. (An example of a globally active window is a scroll bar that can be operated without setting the focus to that client.) If the resource is False, the keyboard input focus is not explicitly set to globally active windows. The default value is True.

fadeNormalIcon (class **FadeNormalIcon**)
If this resource is given a value of True, an icon is grayed out whenever it has been normalized (its window has been opened). The default value is False.

frameBorderWidth (class **FrameBorderWidth**)
This resource specifies the width (in pixels) of a client window frame border without resize handles. The border width includes the 3-D shadows. The default value is 5 pixels.

iconAutoPlace (class **IconAutoPlace**)
This resource indicates whether icons are automatically placed on the screen by mwm, or are placed by the user. Users may specify an initial icon position and may move icons after initial placement; however, mwm will adjust the user-specified position to fit into an invisible grid. When icons are automatically placed, mwm places them into the grid using a scheme set with the iconPlacement resource. If the iconAutoPlace resource has a value of True, then mwm does automatic icon placement. A value of False allows user placement. The default value of this resource is True.

iconBoxGeometry (class **IconBoxGeometry**)
This resource indicates the initial position and size of the icon box. The value of the resource is a standard window geometry string with the following syntax:

[=][*widthxheight*][{+-}*xoffset*{+-}*yoffset*]

If the offsets are not provided, the iconPlacement policy is used to determine the initial placement. The units for width and height are columns and rows. The actual screen size of the icon box window will depend on the iconImageMaximum (size) and iconDecoration resources. The default value for size is (6 * iconWidth + padding) wide by (1 * iconHeight + padding) high. The default value of the location is +0 -0.

iconBoxName (class **IconBoxName**)
This resource specifies the name that is used to look up icon box resources. The default name is iconbox.

iconBoxTitle (class **IconBoxTitle**)
This resource specifies the name that is used in the title area of the icon box frame. The default value is Icons.

iconClick (class **IconClick**)
When this resource is given the value of True, the system menu is posted and left posted when an icon is clicked. The default value is True.

iconDecoration (class **IconDecoration**)
This resource specifies the general icon decoration. The resource value is label"(only the label part is displayed) or image (only the image part is displayed) or label image (both the label and image parts are displayed). A value of activelabel can also be specified to get a label (not truncated to the width of the icon) when the icon is selected. The default icon decoration for icon box icons is that each icon has a label part and an image part (label image). The default icon decoration for stand-alone icons is that each icon has an active label part, a label part and an image part (activelabel label image).

iconImageMaximum (class **IconImageMaximum**)
This resource specifies the maximum size of the icon *image*. The resource value is *widthxheight* (e.g., 64x64). The maximum supported size is 128x128. The default value of this resource is 50x50.

iconImageMinimum (class **IconImageMinimum**)
This resource specifies the minimum size of the icon *image*. The resource value is *widthxheight* (e.g., 32x50). The minimum supported size is 16x16. The default value of this resource is 32x32.

iconPlacement (class **IconPlacement**)
This resource specifies the icon placement scheme to be used. The resource value has

the following syntax:

primary_layout secondary_layout

The layout values are one of the following:

top	Lay the icons out top to bottom.
bottom	Lay the icons out bottom to top.
left	Lay the icons out left to right.
right	Lay the icons out right to left.

A horizontal (vertical) layout value should not be used for both the *primary_layout* and the *secondary_layout* (e.g., don't use top for the *primary_layout* and bottom for the *secondary_layout*). The *primary_layout* indicates whether, when an icon placement is done, the icon is placed in a row or a column and the direction of placement. The *secondary_layout* indicates where to place new rows or columns. For example, top right indicates that icons should be placed top to bottom on the screen and that columns should be added from right to left on the screen. The default placement is left bottom (icons are placed left to right on the screen, with the first row on the bottom of the screen, and new rows added from the bottom of the screen to the top of the screen).

iconPlacementMargin (class **IconPlacementMargin**)
This resource sets the distance between the edge of the screen and the icons that are placed along the edge of the screen. The value should be greater than or equal to 0. A default value (see below) is used if the value specified is invalid. The default value for this resource is equal to the space between icons as they are placed on the screen (this space is based on maximizing the number of icons in each row and column).

interactivePlacement (class **InteractivePlacement**)
This resource controls the initial placement of new windows on the screen. If the value is True, then the pointer shape changes before a new window is placed on the screen to indicate to the user that a position should be selected for the upper-left hand corner of the window. If the value is False, then windows are placed according to the initial window configuration attributes. The default value of this resource is False.

keyBindings (class **KeyBindings**)
This resource identifies the set of key bindings for window management functions. If specified these key bindings *replace* the built-in default bindings. The named set of key bindings is specified in *mwm resource description file*. The default value for this resource is the set of compatible key bindings.

keyboardFocusPolicy (class **KeyboardFocusPolicy**)
If set to pointer, the keyboard focus policy is to have the keyboard focus set to the client window that contains the pointer (the pointer could also be in the client window decoration that mwm adds). If set to explicit, the policy is to have the keyboard focus set to a client window when the user presses button 1 with the pointer on the client window or any part of the associated mwm decoration. The default value for this resource is explicit.

limitResize (class **LimitResize**)
If this resource is True, the user is not allowed to resize a window to greater than the maximum size. The default value for this resource is True.

lowerOnIconify (class **LowerOnIconify**)
If this resource is given the default value of True, a window's icon appears on the bottom of the window stack when the window is minimized (iconified). A value of False places the icon in the stacking order at the same place as its associated window.

maximumMaximumSize (class **MaximumMaximumSize**)
This resource is used to limit the maximum size of a client window as set by the user or client. The resource value is *widthxheight* (e.g., 1024x1024) where the width and height are in pixels. The default value of this resource is twice the screen width and height.

moveThreshold (class **MoveThreshold**)
This resource is used to control the sensitivity of dragging operations that move windows and icons. The value of this resource is the number of pixels that the locator will be moved with a button down before the move operation is initiated. This is used to prevent window/icon movement when a click or double-click is done and there is unintentional pointer movement with the button down. The default value of this resource is 4 (pixels).

passButtons (class **PassButtons**)
This resource indicates whether or not button press events are passed to clients after they are used to do a window manager function in the client context. If the resource value is False, then the button press will not be passed to the client. If the value is True, the button press is passed to the client window. The window manager function is done in either case. The default value for this resource is False.

passSelectButton (class **PassSelectButton**)
This resource indicates whether or not the keyboard input focus selection button press (if keyboardFocusPolicy is explicit) is passed on to the client window or used to do a window management action associated with the window decorations. If the resource value is False then the button press will not be used for any operation other than selecting the window to be the keyboard input focus; if the value is True, the button press is passed to the client window or used to do a window management operation, if appropriate. The keyboard input focus selection is done in either case. The default value for this resource is True.

positionIsFrame (class **PositionIsFrame**)
This resource indicates how client window position information (from the WM_NORMAL_HINTS property and from configuration requests) is to be interpreted. If the resource value is True then the information is interpreted as the position of the mwm client window frame. If the value is False then it is interpreted as being the position of the client area of the window. The default value of this resource is True.

positionOnScreen (class **PositionOnScreen**)
This resource is used to indicate that windows should initially be placed (if possible) so that they are not clipped by the edge of the screen (if the resource value is True). If a window is larger then the size of the screen then at least the upper left corner of the window will be on-screen. If the resource value is False, then windows are placed in the requested position even if totally off-screen. The default value of this resource is True.

quitTimeout (class **QuitTimeout**)
This resource specifies the amount of time (in milliseconds) that mwm will wait for a client to update the WM_COMMAND property after mwm has sent the WM_SAVE_YOURSELF message. This protocol will only be used for those clients that have a WM_SAVE_YOURSELF atom and no WM_DELETE_WINDOW atom in the WM_PROTOCOLS client window property. The default value of this resource is 1000 (ms). (Refer to the f.kill function for additional information.)

resizeBorderWidth (class **ResizeBorderWidth**)
This resource specifies the width (in pixels) of a client window frame border with resize handles. The specified border width includes the 3-D shadows. The default is 10 (pixels).

resizeCursors (class **ResizeCursors**)
This is used to indicate whether the resize cursors are always displayed when the pointer is in the window size border. If True the cursors are shown, otherwise the window manager cursor is shown. The default value is True.

showFeedback (class **ShowFeedback**)
This resource controls when feedback information is displayed. It controls both window position and size feedback during move or resize operations and initial client placement. It also controls window manager message and dialog boxes. The value for this resource is a list of names of the feedback options to be enabled; the names must be separated by

a space. The names of the feedback options are shown below:

Name	Description
all	Show all feedback. (Default value.)
behavior	Confirm behavior switch.
move	Show position during move.
none	Show no feedback.
placement	Show position and size during initial placement.
resize	Show size during resize.
restart	Confirm mwm restart.

The following command line illustrates the syntax for showFeedback:

Mwm*showFeedback: placement resize behavior restart

This resource specification provides feedback for initial client placement and resize, and enables the dialog boxes to confirm the restart and set behavior functions. It disables feedback for the move function.

startupKeyFocus (class **StartupKeyFocus**)
This resource is only available when the keyboard input focus policy is explicit. When given the default value of True, a window gets the keyboard input focus when the window is mapped (i.e., initially managed by the window manager).

transientDecoration (class **TransientDecoration**)
This controls the amount of decoration that Mwm puts on transient windows. The decoration specification is exactly the same as for the **clientDecoration** (client specific) resource. Transient windows are identified by the WM_TRANSIENT_FOR property which is added by the client to indicate a relatively temporary window. The default value for this resource is menu title (i.e., transient windows will have resize borders and a titlebar with a window menu button).

transientFunctions (class **TransientFunctions**)
This resource is used to indicate which window management functions are applicable (or not applicable) to transient windows. The function specification is exactly the same as for the **clientFunctions** (client specific) resource. The default value for this resource is -minimize -maximize.

useIconBox (class **UseIconBox**)
If this resource is given a value of True, icons are placed in an icon box. When an icon box is not used, the icons are placed on the root window (default value).

wMenuButtonClick (class **WMenuButtonClick**)
This resource indicates whether a click of the mouse when the pointer is over the window menu button will post and leave posted the system menu. If the value given this resource is True, then the menu will remain posted. True is the default value for this resource.

wMenuButtonClick2 (class **WMenuButtonClick2**)
When this resource is given the default value of True, a double-click action on the window menu button will do an f.kill function.

CLIENT SPECIFIC RESOURCES

The syntax for specifying *client specific resources* is

Mwm*client_name_or_class***resource_id*

For example, **Mwm*mterm*windowMenu** is used to specify the window menu to be used with mterm clients.

The syntax for specifying *client specific resources* for all classes of clients is

Mwmresource_id***

Specific client specifications take precedence over the specifications for all clients. For example, **Mwm*windowMenu** is used to specify the window menu to be used for all classes of clients that don't have a window menu specified.

The syntax for specifying resource values for windows that have an unknown name and class (i.e. the window does not have a WM_CLASS property associated with it) is

Mwm*defaults*resource_id***

For example, **Mwm*defaults*iconImage** is used to specify the icon image to be used for windows that have an unknown name and class.

The following *client specific resources* can be specified:

Client Specific Resources			
Name	**Class**	**Value Type**	**Default**
clientDecoration	ClientDecoration	string	all
clientFunctions	ClientFunctions	string	all
focusAutoRaise	FocusAutoRaise	T/F	T
iconImage	IconImage	pathname	(image)
iconImageBackground	Background	color	icon background
iconImageBottomShadowColor	Foreground	color	icon bottom shadow
iconImageBottomShadowPixmap	BottomShadowPixmap	color	icon bottom shadow pixmap
iconImageForeground	Foreground	color	icon foreground
iconImageTopShadowColor	Background	color	icon top shadow color
iconImageTopShadowPixmap	TopShadowPixmap	color	icon top shadow pixmap
matteBackground	Background	color	background
matteBottomShadowColor	Foreground	color	bottom shadow color
matteBottomShadowPixmap	BottomShadowPixmap	color	bottom shadow pixmap
matteForeground	Foreground	color	foreground
matteTopShadowColor	Background	color	top shadow color
matteTopShadowPixmap	TopShadowPixmap	color	top shadow pixmap
matteWidth	MatteWidth	pixels	0
maximumClientSize	MaximumClientSize	wxh	fill the screen
useClientIcon	UseClientIcon	T/F	F
windowMenu	WindowMenu	string	string

clientDecoration (class ClientDecoration)

This resource controls the amount of window frame decoration. The resource is specified as a list of decorations to specify their inclusion in the frame. If a decoration is preceded by a minus sign, then that decoration is excluded from the frame. The *sign* of the first item in the list determines the initial amount of decoration. If the sign of the first decoration is minus, then mwm assumes all decorations are present and starts subtracting from that set. If the sign of the first decoration is plus (or not specified), then mwm starts with no decoration and builds up a list from the resource.

Name	Description
all	Include all decorations (default value).
border	Window border.
maximize	Maximize button (includes title bar).
minimize	Minimize button (includes title bar).
none	No decorations.
resizeh	Border resize handles (includes border).
menu	Window menu button (includes title bar).
title	Title bar (includes border).

Examples:

Mwm*XClock*clientDecoration: -resizeh -maximize

This removes the resize handles and maximize button from XClock windows.

Mwm*XClock*clientDecoration: menu minimize border

This does the same thing as above. Note that either **menu** or **minimize** implies **title**.

clientFunctions (class **ClientFunctions**)
This resource is used to indicate which MWM functions are applicable (or not applicable) to the client window. The value for the resource is a list of functions. If the first function in the list has a minus sign in front of it, then MWM starts with all functions and subtracts from that set. If the first function in the list has a plus sign in front of it, then MWM starts with no functions and builds up a list. Each function in the list must be preceded by the appropriate plus or minus sign and be separated from the next function by a space.

The table below lists the functions available for this resource:

Name	Description
all	Include all functions (default value)
none	No functions
resize	f.resize
move	f.move
minimize	f.minimize
maximize	f.maximize
close	f.kill

focusAutoRaise (class **FocusAutoRaise**)
When the value of this resource is True, clients are made completely unobscured when they get the keyboard input focus. If the value is False, the stacking of windows on the display is not changed when a window gets the keyboard input focus. The default value is True.

iconImage (class **IconImage**)
This resource can be used to specify an icon image for a client (e.g., "Mwm*myclock*iconImage"). The resource value is a pathname for a bitmap file. The value of the (client specific) useClientIcon resource is used to determine whether or not user supplied icon images are used instead of client supplied icon images. The default value is to display a built-in window manager icon image.

iconImageBackground (class **Background**)
This resource specifies the background color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon background color (i.e., specified by "Mwm*background or Mwm*icon*background).

iconImageBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow color (i.e., specified by Mwm*icon*bottomShadowColor).

iconImageBottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow Pixmap of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow Pixmap (i.e., specified by Mwm*icon*bottomShadowPixmap).

iconImageForeground (class **Foreground**)
This resource specifies the foreground color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon foreground color (i.e., specified by "Mwm*foreground or Mwm*icon*foreground).

iconImageTopShadowColor (class **Background**)
This resource specifies the top shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow color (i.e., specified by Mwm*icon*topShadowColor).

iconImageTopShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow Pixmap of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow Pixmap (i.e., specified by Mwm*icon*topShadowPixmap).

matteBackground (class **Background**)
This resource specifies the background color of the matte, when **matteWidth** is positive. The default value of this resource is the client background color (i.e., specified by "Mwm*background or Mwm*client*background).

matteBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the matte, when **matteWidth** is positive. The default value of this resource is the client bottom shadow color (i.e., specified by "Mwm*bottomShadowColor or Mwm*client*bottomShadowColor).

matteBottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow Pixmap of the matte, when **matteWidth** is positive. The default value of this resource is the client bottom shadow Pixmap (i.e., specified by "Mwm*bottomShadowPixmap or Mwm*client*bottomShadowPixmap).

matteForeground (class **Foreground**)
This resource specifies the foreground color of the matte, when **matteWidth** is positive. The default value of this resource is the client foreground color (i.e., specified by "Mwm*foreground or Mwm*client*foreground).

matteTopShadowColor (class **Background**)
This resource specifies the top shadow color of the matte, when **matteWidth** is positive. The default value of this resource is the client top shadow color (i.e., specified by "Mwm*topShadowColor or Mwm*client*topShadowColor).

matteTopShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow Pixmap of the matte, when **matteWidth** is positive. The default value of this resource is the client top shadow Pixmap (i.e., specified by "Mwm*topShadowPixmap or Mwm*client*topShadowPixmap).

matteWidth (class **MatteWidth**)
This resource specifies the width of the optional matte. The default value is 0, which effectively disables the matte.

maximumClientSize (class **MaximumClientSize**)
This is a size specification that indicates the client size to be used when an application is maximized. The resource value is specified as *widthxheight*. The width and height are interpreted in the units that the client uses (e.g., for terminal emulators this is generally characters). If this resource is not specified then the maximum size from the WM_NORMAL_HINTS property is used if set. Otherwise the default value is the size

where the client window with window management borders fills the screen. When the maximum client size is not determined by the maximumClientSize resource, the maximumMaximumSize resource value is used as a constraint on the maximum size.

useClientIcon (class **UseClientIcon**)
If the value given for this resource is True, then a client supplied icon image will take precedence over a user supplied icon image. The default value is False, making the user supplied icon image have higher precedence than the client supplied icon image.

windowMenu (class **WindowMenu**)
This resource indicates the name of the menu pane that is posted when the window menu is popped up (usually by pressing button 1 on the window menu button on the client window frame). Menu panes are specified in the *mwm resource description file* file. Window menus can be customized on a client class basis by specifying resources of the form **Mwm*****client_name_or_class**windowMenu** (see the "Mwm Resource Description File Syntax" section in this man page). The default value of this resource is the name of the built-in window menu specification.

RESOURCE DESCRIPTION FILE

The *mwm resource description file* is a supplementary resource file that contains resource descriptions that are referred to by entries in the defaults files (.Xdefaults, app-defaults/Mwm). It contains descriptions of resources that are to be used by mwm, and that cannot be easily encoded in the defaults files (a bitmap file is an analogous type of resource description file). A particular *mwm resource description file* can be selected using the **configFile** resource.

The following types of resources can be described in the *mwm resource description file*:

Buttons	Window manager functions can be bound (associated) with button events.
Keys	Window manager functions can be bound (associated) with key press events.
Menus	Menu panes can be used for the window menu and other menus posted with key bindings and button bindings.

MWM RESOURCE DESCRIPTION FILE SYNTAX

The *mwm resource description file* is a standard text file that contains items of information separated by blanks, tabs, and new lines characters. Blank lines are ignored. Items or characters can be quoted to avoid special interpretation (e.g., the comment character can be quoted to prevent it from being interpreted as the comment character). A quoted item can be contained in double quotes ("). Single characters can be quoted by preceding them by the back-slash character (\). All text from an unquoted # to the end of the line is regarded as a comment and is not interpreted as part of a resource description. If ! is the first character in a line, the line is regarded as a comment. Window manager functions can be accessed with button and key bindings, and with window manager menus. Functions are indicated as part of the specifications for button and key binding sets, and menu panes. The function specification has the following syntax:

function = *function_name* [*function_args*]
function_name = *window manager function*
function_args = {*quoted_item* | *unquoted_item*}

The following functions are supported. If a function is specified that isn't one of the supported functions then it is interpreted by mwm as *f.nop*.

f.beep This function causes a beep.

f.circle_down [icon | window]
This function causes the window or icon that is on the top of the window stack to be put on the bottom of the window stack (so that it is no longer obscuring any

other window or icon). This function affects only those windows and icons that are obscuring other windows and icons, or that are obscured by other windows and icons. Secondary windows (i.e. transient windows) are restacked with their associated primary window. Secondary windows always stay on top of the associated primary window and there can be no other primary windows between the secondary windows and their primary window. If an **icon** function argument is specified, then the function applies only to icons. If a **window** function argument is specified then the function applies only to windows.

f.circle_up [icon | window]
This function raises the window or icon on the bottom of the window stack (so that it is not obscured by any other windows). This function affects only those windows and icons that are obscuring other windows and icons, or that are obscured by other windows and icons. Secondary windows (i.e. transient windows) are restacked with their associated primary window. If an *icon* function argument is specified then the function applies only to icons. If an *window* function argument is specified then the function applies only to windows.

f.exec or **!**
This function causes *command* to be executed (using the value of the *$SHELL* environment variable if it is set, otherwise */bin/sh*). The **!** notation can be used in place of the **f.exec** function name.

f.focus_color
This function sets the colormap focus to a client window. If this function is done in a root context, then the default colormap (setup by the *X Window System* for the screen where mwm is running) is installed and there is no specific client window colormap focus. This function is treated as *f.nop* if *colormapFocusPolicy* is not explicit.

f.focus_key
This function sets the keyboard input focus to a client window or icon. This function is treated as *f.nop* if *keyboardFocusPolicy* is not explicit or the function is executed in a root context.

f.kill If the WM_DELETE_WINDOW protocol is set up, the client is sent a client message event indicating that the client window should be deleted. If the WM_SAVE_YOURSELF protocol is set up and the WM_DELETE_WINDOW protocol is not set up, the client is sent a client message event indicating that the client needs to prepare to be terminated. If the client does not have the WM_DELETE_WINDOW or WM_SAVE_YOURSELF protocol set up, this function causes a client's X connection to be terminated (usually resulting in termination of the client). Refer to the description of the *quitTimeout* resource and the *WM_PROTOCOLS* property.

f.lower [*-client*]
This function lowers a client window to the bottom of the window stack (where it obscures no other window). Secondary windows (i.e. transient windows) are restacked with their associated primary window. The *client* argument indicates the name or class of a client to lower. If the *client* argument is not specified then the context that the function was invoked in indicates the window or icon to lower.

f.maximize
This function causes a client window to be displayed with its maximum size.

f.menu This function associates a cascading (pull-right) menu with a menu pane entry or a menu with a button or key binding. The *menu_name* function argument identifies the menu to be used.

f.minimize
This function causes a client window to be minimized (iconified). When a window is minimized with no icon box in use, and if the **lowerOnIconify** resource

has the value True (the default), the icon is placed on the bottom of the window stack (such that it obscures no other window). If an icon box is used, then the client's icon changes to its iconified form inside the icon box. Secondary windows (i.e. transient windows) are minimized with their associated primary window. There is only one icon for a primary window and all its secondary windows.

f.move This function allows a client window or icon to be moved interactively.

f.next_cmap
This function installs the next colormap in the list of colormaps for the window with the colormap focus.

f.next_key [icon | window | transient]
This function sets the keyboard input focus to the next window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as *f.nop* if *keyboardFocusPolicy* is not explicit. The keyboard input focus is only moved to windows that do not have an associated secondary window that is application modal. If the **transient** argument is specified, then transient (secondary) windows are traversed (otherwise, if only **window** is specified, traversal is done only to the last focused window in a transient group). If an **icon** function argument is specified, then the function applies only to icons. If a **window** function argument is specified, then the function applies only to windows.

f.nop This function does nothing.

f.normalize
This function causes a client window to be displayed with its normal size. Secondary windows (i.e. transient windows) are placed in their normal state along with their associated primary window.

f.pack_icons
This function is used to relayout icons (based on the layout policy being used) on the root window or in the icon box. In general this causes icons to be "packed" into the icon grid.

f.pass_keys
This function is used to enable/disable (toggle) processing of key bindings for window manager functions. When it disables key binding processing all keys are passed on to the window with the keyboard input focus and no window manager functions are invoked. If the *f.pass_keys* function is invoked with a key binding to disable key binding processing the same key binding can be used to enable key binding processing.

f.post_wmenu
This function is used to post the window menu. If a key is used to post the window menu and a window menu button is present, the window menu is automatically placed with its top-left corner at the bottom-left corner of the window menu button for the client window. If no window menu button is present, the window menu is placed at the top-left corner of the client window.

f.prev_cmap
This function installs the previous colormap in the list of colormaps for the window with the colormap focus.

f.prev_key [icon | window | transient]
This function sets the keyboard input focus to the previous window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as *f.nop* if *keyboardFocusPolicy* is not explicit. The keyboard input focus is only moved to windows that do not have an associated secondary window that is application modal. If the *transient* argument is specified, then transient (secondary) windows are traversed (otherwise, if only *window* is specified, traversal is done only to the last focused window in a transient group). If an *icon* function argument is

specified then the function applies only to icons. If an *window* function argument is specified then the function applies only to windows.

f.quit_mwm
This function terminates mwm (but NOT the X window system).

f.raise [*-client*]
This function raises a client window to the top of the window stack (where it is obscured by no other window). Secondary windows (i.e. transient windows) are restacked with their associated primary window. The *client* argument indicates the name or class of a client to raise. If the *client* argument is not specified then the context that the function was invoked in indicates the window or icon to raise.

f.raise_lower
This function raises a client window to the top of the window stack if it is partially obscured by another window, otherwise it lowers the window to the bottom of the window stack. Secondary windows (i.e. transient windows) are restacked with their associated primary window.

f.refresh
This function causes all windows to be redrawn.

f.refresh_win
This function causes a client window to be redrawn.

f.resize This function allows a client window to be interactively resized.

f.restart
This function causes mwm to be restarted (effectively terminated and re-executed).

f.send_msg *message_number*
This function sends a client message of the type _MOTIF_WM_MESSAGES with the *message_type* indicated by the *message_number* function argument. The client message will only be sent if *message_number* is included in the client's _MOTIF_WM_MESSAGES property. A menu item label is grayed out if the menu item is used to do *f.send_msg* of a message that is not included in the client's _MOTIF_WM_MESSAGES property.

f.separator
This function causes a menu separator to be put in the menu pane at the specified location (the label is ignored).

f.set_behavior
This function causes the window manager to restart with the default behavior (if a custom behavior is configured) or a custom behavior (if a default behavior is configured).

f.title This function inserts a title in the menu pane at the specified location.

Each function may be constrained as to which resource types can specify the function (e.g., menu pane) and also what context the function can be used in (e.g., the function is done to the selected client window). Function contexts are

root	No client window or icon has been selected as an object for the function.
window	A client window has been selected as an object for the function. This includes the window's title bar and frame. Some functions are applied only when the window is in its normalized state (e.g., f.maximize) or its maximized state (e.g., f.normalize).
icon	An icon has been selected as an object for the function.

If a function is specified in a type of resource where it is not supported or is invoked in a context that does not apply then the function is treated as *f.nop*. The following table indicates the resource types and function contexts in which window manager functions apply.

Series 300 and 800 Only

Function	Contexts	Resources
f.beep	root,icon,window	button,key,menu
f.circle_down	root,icon,window	button,key,menu
f.circle_up	root,icon,window	button,key,menu
f.exec	root,icon,window	button,key,menu
f.focus_color	root,icon,window	button,key,menu
f.focus_key	root,icon,window	button,key,menu
f.kill	icon,window	button,key,menu
f.lower	root,icon,window	button,key,menu
f.maximize	icon,window(normal)	button,key,menu
f.menu	root,icon,window	button,key,menu
f.minimize	window	button,key,menu
f.move	icon,window	button,key,menu
f.next_cmap	root,icon,window	button,key,menu
f.next_key	root,icon,window	button,key,menu
f.nop	root,icon,window	button,key,menu
f.normalize	icon,window(maximized)	button,key,menu
f.pack_icons	root,icon,window	button,key,menu
f.pass_keys	root,icon,window	button,key,menu
f.post_wmenu	root,icon,window	button,key
f.prev_cmap	root,icon,window	button,key,menu
f.prev_key	root,icon,window	button,key,menu
f.quit_mwm	root	button,key,menu
f.raise	root,icon,window	button,key,menu
f.raise_lower	icon,window	button,key,menu
f.refresh	root,icon,window	button,key,menu
f.refresh_win	window	button,key,menu
f.resize	window	button,key,menu
f.restart	root	button,key,menu
f.send_msg	icon,window	button,key,menu
f.separator	root,icon,window	menu
f.set_behavior	root,icon,window	button,key,menu
f.title	root,icon,window	menu

WINDOW MANAGER EVENT SPECIFICATION

Events are indicated as part of the specifications for button and key binding sets, and menu panes.

Button events have the following syntax:

button = [*modifier_list*]<*button_event_name*>
modifier_list = *modifier_name* {*modifier_name*}

All modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the button event occurs). The following table indicates the values that can be used for *modifier_name*. The [Alt] key is frequently labeled [Extend] or [Meta]. Alt and Meta can be used interchangeably in event specification.

Series 300 and 800 Only

Modifier	Description
Ctrl	Control Key
Shift	Shift Key
Alt	Alt/Meta Key
Meta	Meta/Alt Key
Lock	Lock Key
Mod1	Modifier1
Mod2	Modifier2
Mod3	Modifier3
Mod4	Modifier4
Mod5	Modifier5

The following table indicates the values that can be used for *button_event_name*.

Button	Description
Btn1Down	Button 1 Press
Btn1Up	Button 1 Release
Btn1Click	Button 1 Press and Release
Btn1Click2	Button 1 Double Click
Btn2Down	Button 2 Press
Btn2Up	Button 2 Release
Btn2Click	Button 2 Press and Release
Btn2Click2	Button 2 Double Click
Btn3Down	Button 3 Press
Btn3Up	Button 3 Release
Btn3Click	Button 3 Press and Release
Btn3Click2	Button 3 Double Click
Btn4Down	Button 4 Press
Btn4Up	Button 4 Release
Btn4Click	Button 4 Press and Release
Btn4Click2	Button 4 Double Click
Btn5Down	Button 5 Press
Btn5Up	Button 5 Release
Btn5Click	Button 5 Press and Release
Btn5Click2	Button 5 Double Click

Key events that are used by the window manager for menu mnemonics and for binding to window manager functions are single key presses; key releases are ignored. Key events have the following syntax:

key = [*modifier_list*]<**Key**>*key_name*
modifier_list = *modifier_name* {*modifier_name*}

All modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the key event occurs). Modifiers for keys are the same as those that apply to buttons. The *key_name* is an X11 keysym name. Keysym names can be found in the keysymdef.h file (remove the *XK_* prefix).

BUTTON BINDINGS

The **buttonBindings** resource value is the name of a set of button bindings that are used to configure window manager behavior. A window manager function can be done when a button press occurs with the pointer over a framed client window, an icon or the root window. The context for indicating where the button press applies is also the context for invoking the window manager function when the button press is done (significant for functions that are context sensitive).

The button binding syntax is

```
Buttons bindings_set_name
{
  button   context   function
  button   context   function
                .
                .
  button   context   function
}
```

The syntax for the *context* specification is

```
context =   object[|context]
object =    root | icon | window | title | frame | border | app
```

The context specification indicates where the pointer must be for the button binding to be effective. For example, a context of **window** indicates that the pointer must be over a client window or window management frame for the button binding to be effective. The **frame** context is for the window management frame around a client window (including the border and titlebar), the **border** context is for the border part of the window management frame (not including the titlebar), the **title** context is for the title area of the window management frame, and the **app** context is for the application window (not including the window management frame).

If an *f.nop* function is specified for a button binding, the button binding will not be done.

KEY BINDINGS

The **keyBindings** resource value is the name of a set of key bindings that are used to configure window manager behavior. A window manager function can be done when a particular key is pressed. The context in which the key binding applies is indicated in the key binding specification. The valid contexts are the same as those that apply to button bindings.

The key binding syntax is

```
Keys bindings_set_name
{
  key   context   function
  key   context   function
              .
              .
  key   context   function
}
```

If an *f.nop* function is specified for a key binding, the key binding will not be done. If an *f.post_wmenu* or *f.menu* function is bound to a key, mwm will automatically use the same key for removing the menu from the screen after it has been popped up.

The *context* specification syntax is the same as for button bindings. For key bindings, the **frame**, **title**, **border**, and **app** contexts are equivalent to the **window** context. The context for a key event is the window or icon that has the keyboard input focus (**root** if no window or icon has the keyboard input focus).

MENU PANES

Menus can be popped up using the *f.post_wmenu* and *f.menu* window manager functions. The context for window manager functions that are done from a menu is *root*, *icon* or *window* depending on how the menu was popped up. In the case of the *window* menu or menus popped up with a key binding, the location of the keyboard input focus indicates the context. For menus popped up using a button binding, the context of the button binding is the context of the menu.

The menu pane specification syntax is

```
Menu menu_name
{
  label [mnemonic] [accelerator] function
  label [mnemonic] [accelerator] function
        .
        .
  label [mnemonic] [accelerator] function
}
```

Each line in the *Menu* specification identifies the label for a menu item and the function to be done if the menu item is selected. Optionally a menu button mnemonic and a menu button keyboard accelerator may be specified. Mnemonics are functional only when the menu is posted and keyboard traversal applies.

The *label* may be a string or a bitmap file. The label specification has the following syntax:

```
label =        text | bitmap_file
bitmap_file =  @file_name
text =         quoted_item | unquoted_item
```

The string encoding for labels must be compatible with the menu font that is used. Labels are greyed out for menu items that do the *f.nop* function or an invalid function or a function that doesn't apply in the current context.

A *mnemonic* specification has the following syntax

```
mnemonic =  _character
```

The first matching *character* in the label is underlined. If there is no matching *character* in the label, no mnemonic is registered with the window manager for that label. Although the *character* must exactly match a character in the label, the mnemonic will not execute if any modifier (such as Shift) is pressed with the character key.

The *accelerator* specification is a key event specification with the same syntax as is used for key bindings to window manager functions.

ENVIRONMENT

Mwm uses the environment variable **$HOME** specifying the user's home directory.

FILES

/usr/lib/X11/system.mwmrc
/usr/lib/X11/app-defaults/Mwm
$HOME/.Xdefaults
$HOME/.mwmrc

COPYRIGHT

RELATED INFORMATION

X(1), **VendorShell(3X)**, and **XmInstallImage(3X)**.

Series 300 and 800 Only

NAME

resize - reset shell parameters to reflect the current size of a window

SYNOPSIS

resize [-option ...]

DESCRIPTION

Resize prints on its standard output the commands for setting $TERM, $LINES, and $COLUMNS for a shell to reflect the current size of its window. The $SHELL environment variable is used to determine the shell for which to form the commands. The $TERM environment variable is used to determine the escape sequences to be used to determine the window size. Both of these can be overridden by command line options.

Resize is never executed directly, but should be run via *eval*(1) similar to *tset*(1) to cause the shell to execute the commands. For example, the following functions will reset the environment of the current shell for *sh*(1) and *ksh*(1):

```
xs()    { eval 'resize'; }
xrs()   { eval 'resize -s $@'; }
```

An equivalent for *csh*(1) is:

```
alias xs  'set noglob; eval `resize`'
alias xrs 'set noglob; eval `resize -s \!\*`'
```

OPTIONS

The *resize* program accepts the following options listed below:

-c This option indicates that *resize* should format its commands for *csh*(1).

-h This option indicates that *resize* should use Hewlett Packard terminal escape sequences to obtain the terminal's new window size.

-s [row col]
This option indicates that *resize* should use Sun escape sequences to obtain the terminal's new window size. In this mode of operation, a new row and column size may be specified on the command line.

-u This option indicates that *resize* should format its commands for *sh*(1) or *ksh*(1).

-x This option indicates that *resize* should use VT102 escape sequences to obtain the terminal's new window size.

FILES

$HOME/.profile	*sh*(1) and *ksh*(1) user's functions for *resize*.
$HOME/.cshrc	*csh*(1) user's alias for *resize*.

NOTES

"-s" must be the last option on the command line when specified.

There should be some global notion of display size; termcap and terminfo need to be rethought in the context of window systems.

ORIGINS

MIT Distribution

SEE ALSO

sh(1), ksh(1), csh(1), eval(1), hpterm(1), tset(1), xterm(1)

NAME

rgb - X Window System color database creator.

SYNOPSIS

rgb [*filename*] [<*input filename*]

DESCRIPTION

rgb creates a data base used by the X window system server for its colors. Stdin is used as its input and must be in the format of:

0-255 0-255 0-255 colorname

For example:

```
  0   0   0   black
  0 128   0   green
255 255 255   white
```

rgb stands for *r*ed-*g*reen-*b*lue. Each element can have no intensity (0) to full intensity (255). How the elements are combined determines the actual color. The name given to the color can be descriptive or fanciful.

In other words, the sequence:

```
0   0 128
```

can be given the name 'blue' or 'unicorn blue'. There can also be two (or more) entries with the same element numbers or names.

ARGUMENTS

filename If filename is given, *rgb* produces two files; *filename.dir* and *filename.pag*. Otherwise the default filename is */usr/lib/X11/rgb*.

ORIGIN

MIT Distribution

SEE ALSO

X(1)

Series 300 and 800 Only

NAME

sb2xwd - translate Starbase bitmap to xwd bitmap format

SYNOPSIS

sb2xwd

DESCRIPTION

This command translates a bitmap file created by one of the Starbase bitmap-to-file procedures into an *XWD* format file. The *XWD* format is defined by the *xwd(1)* and *xwud(1)* X window dump utility programs. The Starbase bitmap file format is described in *bitmapfile(4)*. Translation is done from standard input to standard output.

Starbase bitmaps created in *pixel-major* format will be translated into *ZPixmap* format xwd bitmaps. *Plane-major full-depth* Starbase bitmaps are translated into *XYPixmap* format xwd bitmaps.

XWD bitmaps produced by *sb2xwd* will be of the *DirectColor* visual class if the Starbase bitmap's colormap mode is *CMAP_FULL*. Other multiplane Starbase bitmaps having colormap modes of *CMAP_NORMAL* or *CMAP_MONOTONIC* will result in *PseudoColor* XWD bitmaps. Single plane Starbase bitmaps are converted to *GreyScale* XWD bitmaps.

OPTIONS

none

EXAMPLES

sb2xwd < sbfile > xwdfile
Translates the Starbase image in *sbfile* to XWD format and places the result in *xwdfile.*

sb2xwd < sbimage | xwud
Translates the image in *sbimage,* piping the results to *xwud* for display.

RESTRICTIONS

sb2xwd accepts only *full-depth* Starbase bitmaps.

Starbase bitmaps must be 1-8, 12, or 24 planes deep. Bitmaps of depth 1-8 must have either CMAP_NORMAL or CMAP_MONOTONIC colormap modes. Bitmaps of depths 12 or 24 must have the CMAP_FULL colormap mode.

A 12 plane bitmap must be stored in three banks and have a display enable mask of 0x0F or 0xF0.

ORIGIN

Hewlett-Packard GTD

SEE ALSO

xwd(1), xwud(1), bitmapfile(4).

Starbase Graphics Techniques, HP-UX Concepts and Tutorials, chapters on "Color" and "Storing and Printing Images".

NAME

uwm - a window manager for X

SYNTAX

uwm [options]

DESCRIPTION

The *uwm* program is a window manager for X.

When *uwm* is invoked, it searches a predefined search path to locate any *uwm* startup files. If no startup files exist, *uwm* initializes its built-in default file.

If startup files exist in any of the following locations, it adds the variables to the default variables. In the case of contention, the variables in the last file found override previous specifications. Files in the *uwm* search path are:

/usr/lib/X11/uwm/system.uwmrc
$HOME/.uwmrc
-f *filename*

To use only the settings defined in a single startup file, include the variables **resetbindings**, **resetmenus**, **resetvariables** at the top of that specific startup file.

OPTIONS

-f *filename*
Names an alternate file as a *uwm* startup file.

-display *display*
Specifies the display to use; see *X(1)*.

STARTUP FILE VARIABLES

Variables are typically entered first, at the top of the startup file. By convention, **resetbindings**, **resetmenus**, and **resetvariables** head the list.

autoselect/noautoselect
places the menu cursor in the first menu item. If unspecified, the menu cursor is placed in the menu header when the menu is displayed.

background=*color*
specifies the default background color for popup sizing windows, menus, and icons. The default is to use the WhitePixel for the current screen.

bordercolor=*color*
specifies the default border color for popup sizing windows, menus, and icons. The default is to use the BlackPixel for the current screen.

borderwidth=*pixels*
specifies the default width in pixels for borders around popup sizing windows. The default is 2.

delta=*pixels* indicates the number of pixels the cursor is moved before the action is interpreted by the window manager as a command. (Also refer to the **delta** mouse action.)

foreground=*color*
specifies the default foreground color for popup sizing windows, menus, and icons. The default is to use the BlackPixel for the current screen.

freeze/nofreeze locks all other client applications out of the server during certain window manager tasks, such as move and resize.

grid/nogrid displays a finely-ruled grid to help you position an icon or window during resize or move operations.

hiconpad=*pixels* indicates the number of pixels to pad an icon horizontally. The default is five pixels.

hmenupad=*pixels*
indicates the amount of space in pixels that each menu item is padded to the left

and to the right of the text.

iborderwidth=*pixels*
indicates the width in pixels of the border surrounding icons.

iconfont=*fontname*
names the font that is displayed within icons. Font names for a given server can be found in */usr/lib/X11/fonts*.

maxcolors=*number*
limits the number of colors the window manager can use in a given invocation. If set to zero, or not specified, *uwm* assumes no limit to the number of colors it can take from the color map. **maxcolors** counts colors as they are included in the file.

mborderwidth=*pixels*
indicates the width in pixels of the border surrounding menus.

menufont=*fontname*
names the font that is displayed within menus. Font names for a given server can be found in */usr/lib/X11/fonts*.

normali/nonormali
places icons created with **f.newiconify** within the root window, even if it is placed partially off the screen. With **nonormali** the icon is placed exactly where the cursor leaves it.

normalw/nonormalw
places window created with **f.newiconify** within the root window, even if it is placed partially off the screen. With **nonormalw** the window is placed exactly where the cursor leaves it.

push=*number* moves a window *number* pixels or 1/*number* times the size of the window, depending on whether **pushabsolute** or **pushrelative** is specified. Use this variable in conjunction with **f.pushup, f.pushdown, f.pushright**, or **f.pushleft**.

pushabsolute/pushrelative
pushabsolute indicates that the number entered with push is equivalent to pixels. When an f.push (left, right, up, or down) function is called, the window is moved exactly that number of pixels.

pushrelative indicates that the number entered with the push variable represents a relative number. When an f.push function is called, the window is invisibly divided into the number of parts you entered with the push variable, and the window is moved one part.

resetbindings, **resetmenus**, and **resetvariables**
resets all previous function bindings, menus, and variable entries, specified in any startup file in the *uwm* search path, including those in the default environment. By convention, these variables are entered first in the startup file.

resizefont=*fontname*
identifies the font of the indicator that displays dimensions in the corner of the window as you resize windows. Font names for a given server can be found in */usr/lib/X11/fonts*.

resizerelative/noresizerelative
indicates whether or not resize operations should be done relative to moving edge or edges. By default, the dynamic rectangle uses the actual pointer location to define the new size.

reverse/noreverse
defines the display as black characters on a white background for the window manager windows and icons.

viconpad=*pixels* indicates the number of pixels to pad an icon vertically. Default is five pixels.

vmenupad=*pixels*
indicates the amount of space in pixels that the menu is padded above and below the text.

volume=*number* increases or decreases the base level volume set by the *xset(1)* command. Enter an integer from 0 to 7, 7 being the loudest.

zap/nozap causes ghost lines to follow the window or icon from its previous default location to its new location during a move, resize or iconify operation.

BINDING SYNTAX

function=[*control key(s)*]:[*context*]:*mouse events*:"*menu name* "

Function and mouse events are required input. Menu name is required with the *f.menu* function definition only.

Function

f.beep emits a beep from the keyboard. Loudness is determined by the volume variable.

f.circledown causes the top window that is obscuring another window to drop to the bottom of the stack of windows.

f.circleup exposes the lowest window that is obscured by other windows.

f.continue releases the window server display action after you stop action with the **f.pause** function.

f.exit causes the window manager application to quit cleanly.

f.focus directs all keyboard input to the selected window. To reset the focus to all windows, invoke *f.focus* on the root window.

f.iconify when implemented from a window, this function converts the window to its respective icon. When implemented from an icon, f.iconify converts the icon to its respective window.

f.kill kills the client that created a window.

f.lower lowers a window that is obstructing a window below it.

f.menu invokes a menu. Enclose 'menu name' in quotes if it contains blank characters or parentheses.

f.menu=[*control key(s)*]:[*context*]:*mouse events*:"*menu name* "

f.move moves a window or icon to a new location, which becomes the default location.

f.moveopaque moves a window or icon to a new location. When using this function, the entire window or icon is moved to the new screen location. The grid effect is not used with this function.

f.newiconify allows you to create a window or icon and then position the window or icon in a new default location on the screen.

f.pause temporarily stops all display action. To release the screen and immediately update all windows, use the **f.continue** function.

f.pushdown moves a window down. The distance of the push is determined by the push variables.

f.pushleft moves a window to the left. The distance of the push is determined by the push variables.

f.pushright moves a window to the right. The distance of the push is determined by the push variables.

f.pushup moves a window up. The distance of the push is determined by the push variables.

f.raise raises a window that is being obstructed by a window above it.

f.refresh results in exposure events being sent to the window server clients for all unobscured or partially obscured windows. The windows will not refresh correctly if the exposure events are not handled properly.

f.resize resizes an existing window. Note that some clients, notably editors, react unpredictably if you resize the window while the client is running.

f.restart causes the window manager application to restart, retracing the *uwm* search path and initializing the variables it finds.

Control Keys

By default, the window manager uses meta as its control key. It can also use ctrl, shift, lock, or null (no control key). Control keys must be entered in lower case, and can be abbreviated as: c, l, m, s for ctrl, lock, meta, and shift, respectively.

You can bind one or more, or no control keys to a function. Use the bar (|) character to combine control keys.

Note that client applications other than the window manager use the shift as a control key. If you bind the shift key to a window manager function, you can not use other client applications that require this key.

Context

The context refers to the screen location of the cursor when a command is initiated. When you include a context entry in a binding, the cursor must be in that context or the function will not be activated. The window manager recognizes the following four contexts: icon, window, root, (null).

The root context refers to the root, or background window, A (null) context is indicated when the context field is left blank, and allows a function to be invoked from any screen location. Combine contexts using the bar (|) character.

Mouse Buttons

Any of the following mouse buttons are accepted in lower case and can be abbreviated as l, m, or r, respectively: left, middle, right.

With the specific button, you must identify the action of that button. Mouse actions can be:

down function occurs when the specified button is pressed down.

up function occurs when the specified button is released.

delta indicates that the mouse must be moved the number of pixels specified with the delta variable before the specified function is invoked. The mouse can be moved in any direction to satisfy the delta requirement.

MENU DEFINITION

After binding a set of function keys and a menu name to **f.menu**, you must define the menu to be invoked, using the following syntax:

```
menu = " menu name " {
"item name" : "action"
    .
    .
    .
}
```

Enter the menu name exactly the way it is entered with the **f.menu** function or the window manager will not recognize the link. If the menu name contains blank strings, tabs or parentheses, it must be quoted here and in the f.menu function entry. You can enter as many menu items as your screen is long. You cannot scroll within menus.

Any menu entry that contains quotes, special characters, parentheses, tabs, or strings of blanks must be enclosed in double quotes. Follow the item name by a colon (:).

Menu Action

Window manager functions

Any function previously described. E.g., **f.move** or **f.iconify**.

Shell commands

Begin with an exclamation point (!) and set to run in background. You cannot include a new line character within a shell command.

Text strings

Text strings are placed in the window server's cut buffer.

Strings starting with an up arrow (^) will have a new line character appended to the string after the up arrow (^) has been stripped from it.

Strings starting with a bar character (|) will be copied as is after the bar character (|) has been stripped.

Color Menus

Use the following syntax to add color to menus:

menu = "*menu name*" (*color1:color2:color3:color4*) {
"*item name*" : (*color5 :color6*) : " *action* "
.
.
.
}

color1 Foreground color of the header.

color2 Background color of the header.

color3 Foreground color of the highlighter, the horizontal band of color that moves with the cursor within the menu.

color4 Background color of the highlighter.

color5 Foreground color for the individual menu item.

color6 Background color for the individual menu item.

Color Defaults

Colors default to the colors of the root window under any of the following conditions:

1) If you run out of color map entries, either before or during an invocation of *uwm*.

2) If you specify a foreground or background color that does not exist in the RGB color database of the server (see */usr/lib/X11/rgb.txt* for a sample) both the foreground and background colors default to the root window colors.

3) If you omit a foreground or background color, both the foreground and background colors default to the root window colors.

4) If the total number of colors specified in the startup file exceeds the number specified in the *maxcolors* variable.

5) If you specify no colors in the startup file.

Customizing Icon Names

Icon names may be editted by placing the pointer inside the icon and typing. The Backspace, Rubout and Delete keys may be used to remove a character from the end of a line and Control-U may be used to delete the whole name.

EXAMPLES

The following sample startup file shows the default window manager options:

```
# Global variables
#
resetbindings;resetvariables;resetmenus
autoselect
delta=25
```

```
freeze
grid
hiconpad=5
hmenupad=6
iconfont=oldeng
menufont=timrom12b
resizefont=9x15
viconpad=5
vmenupad=3
volume=7
#
# Mouse button/key maps
#
# FUNCTION   KEYS CONTEXT BUTTON   MENU(if any)
# ========  ==== ======= ======  ============
f.menu =     meta :   :left down  :"WINDOW OPS"
f.menu =     meta :   :middle down :"EXTENDED WINDOW OPS"
f.move =     meta :w|i :right down
f.circleup =  meta :root :right down
#
# Menu specifications
#
menu = "WINDOW OPS" {
"(De)Iconify":   f.iconify
Move:            f.move
Resize:          f.resize
Lower:           f.lower
Raise:           f.raise
}

menu = "EXTENDED WINDOW OPS" {
Create Window:                !"xterm &"
Iconify at New Position:  f.lowericonify
Focus Keyboard on Window:     f.focus
Freeze All Windows:           f.pause
Unfreeze All Windows:         f.continue
Circulate Windows Up:         f.circleup
Circulate Windows Down:              f.circledown
}
```

RESTRICTIONS

The color specifications have no effect on a monochrome system.

ENVIRONMENT

DISPLAY - the default host and display number.

FILES

/usr/lib/X11/uwm/system.uwmrc
$HOME/.uwmrc

SEE ALSO

X(1), Xserver(1), xset(1), xlsfonts(1)

COPYRIGHT

ORIGIN

M. Gancarz, DEC Ultrix Engineering Group, Merrimack, New Hampshire, using some algorithms originally by Bob Scheifler, MIT Laboratory for Computer Science.

Series 300 and 800 Only

NAME

X - a portable, network transparent window system

SYNOPSIS

X is a network transparent window system developed at MIT which runs under a wide variety of operating systems. Hewlett-Packard supports the X Window System under the Series 300 HP-UX 6.2 or higher, and the Series 800 HP-UX 3.0 or higher.

THE OFFICIAL NAMES

The official names of the software described herein are:

X
X Window System
X Version 11
X Window System, Version 11
X11

Note that the phrases X.11, X-11, X Windows or any permutation thereof, are explicitly excluded from this list and should not be used to describe the X Window System (window system should be thought of as one word).

X Window System is a trademark of the Massachusetts Institute of Technology.

DESCRIPTION

X window system servers run on computers with bitmap displays. The server distributes user input to, and accepts output requests from various client programs through a variety of different interprocess communication channels. Although the most common case is for the client programs to be running on the same machine as the server, clients can be run transparently from other machines (including machines with different architectures and operating systems) as well.

X supports overlapping hierarchical subwindows and text and graphics operations, on both monochrome and color displays. For a full explanation of functions, see the *Programming With Xlib* manual, the *Programming With the HP X Widgets* manual, and the *Programming With the Xt Intrinsics* manual.

The core *X* protocol provides mechanism, not policy. Windows are manipulated (including moving, resizing and iconifying) not by the server itself, but by a separate program called a "window manager" of your choosing. This program is simply another client and requires no special privileges. If you don't like the one that is supplied you can write your own.

The number of programs that use *X* is growing rapidly. Of particular interest are: two terminal emulators (*hpterm(1)* and *xterm(1)*), window managers (*mwm(1)*, *hpwm(1)*, *uwm(1)*), a bitmap editor (*bitmap(1)*), an access control program (*xhost(1)*), user preference setting programs (*xset(1)*, *xsetroot(1)*, and *xmodmap(1)*), a load monitor (*xload(1)*), clock (*xclock(1)*), a font displayer (*xfd(1)*), and various demos.

DISPLAY SPECIFICATION

When you first start the window system, the environment variable DISPLAY will be set (if it hasn't already been set to something) to local:0.0, in order to take advantage of local interprocess communication (IPC) mechanisms. By convention, servers on a particular machine are numbered starting with zero. The following connection protocols are supported:

TCP/IP

DISPLAY should be set to "*host:display.screen*" where *host* is the symbolic name of the machine (e.g. expo), *display* is the number of the display (usually 0), and *screen* is the number of the screen. The *screen* and preceding period are optional, with the default value being zero (0). Full Internet domain names (e.g. expo.lcs.mit.edu) are allowed for the host name.

IPC Mechanisms

DISPLAY should be set to "local:*display.screen*", where *display* is the display number and *screen* is the screen number; *screen* and the preceding period are optional, with the default value being zero (0).

Most programs accept a command line argument of the form "-display *display*" that can be used to override the DISPLAY environment variable.

GEOMETRY SPECIFICATION

One of the advantages of using window systems over hardwired terminals is that applications don't have to be restricted to a particular size or location on the screen. Although the layout of windows on a display is controlled by the window manager that the user is running, most applications accept a command line argument that is treated as the preferred size and location for this particular application's window.

This argument, usually specified as "-geometry WxH+X+Y," indicates that the window should have a width of W and height of H (usually measured in pixels or characters, depending on the application), and the upper left corner X pixels to the right and Y pixels below the upper left corner of the screen (origin (0,0)). "WxH" can be omitted to obtain the default application size, or "+X+Y" can be omitted to obtain the default application position (which is usually then left up to the window manager or user to choose). The X and Y values may be negative to position the window off the screen. In addition, if minus signs are used instead of plus signs (e.g. WxH-X-Y), then (X,Y) represents the location of the lower right hand corner of the window relative to the lower right hand corner of the screen.

By combining plus and minus signs, the window may be placed relative to any of the four corners of the screen. For example:

555x333+11+22
This will request a window 555 pixels wide and 333 pixels tall, with the upper left corner located at (11,22).

300x200-0+0
This will request a window measuring 300 by 200 pixels in the upper right hand corner of the screen.

48x48-5-10
This will request a window measuring 48 by 48 pixels whose lower right hand corner is 5 pixels off the right edge of the screen and 10 pixels off the bottom edge.

COMMAND LINE ARGUMENTS

Most X programs attempt to use a common set of names for their command line arguments. The X Toolkit automatically handles the following arguments:

-bg *color*, **-background** *color*
Either option specifies the color to use for the window background.

-bd *color*, **-bordercolor** *color*
Either option specifies the color to use for the window border.

-bw *number*, **-borderwidth** *number*
Either option specifies the width in pixels of the window border.

-display *display*
This option specifies the name of the X server to use.

-fg *color*, **-foreground** *color*
Either option specifies the color to use for text or graphics.

-fn *font*, **-font** *font*
Either option specifies the font to use for displaying text. The font is in one of the directories listed when executing xset q.

-geometry *geometry*
This option specifies the initial size and location of the window.

-iconic
This option indicates that application should start out in an iconic state. Note that how this state is represented is controlled by the window manager that the user is running.

-name
This option specifies the name under which resources for the application should be

found. This option is useful in shell aliases to distinguish between invocations of an application, without resorting to creating links to alter the executable file name.

-rv, -reverse
Either option indicates that the program should simulate reverse video if possible, often by swapping the foreground and background colors. Not all programs honor this or implement it correctly. It is usually only used on monochrome displays.

+rv
This option indicates that the program should not simulate reverse video. This is used to override any defaults since reverse video doesn't always work properly.

-synchronous
This option indicates that requests to the X server should be sent synchronously, instead of asynchronously. Since *Xlib* normally buffers requests to the server, errors do not necessarily get reported immediately after they occur. This option turns off the buffering so that the application can be debugged. It should never be used with a working program.

-title *string*
This option specifies the title to be used for this window. This information is used by some window managers to provide some sort of header identifying the window.

-xrm *resourcestring*
This option specifies a resource name and value to override any defaults. It is also very useful for setting resources that don't have explicit command line arguments.

RESOURCES
To make the tailoring of applications to personal preferences easier, X supports several mechanisms for storing default values for program resources (e.g. background color, window title, etc.) Resources are specified as strings of the form *"name*subname*subsubname...: value"* (see the *Xlib* manual section *Using the Resource Manager* for more details) that are loaded into a client when it starts up. The *Xlib* routine *XGetDefault(3X)* and the resource utilities within the X Toolkit obtain resources from the following sources:

RESOURCE_MANAGER root window property
Any global resources that should be available to clients on all machines should be stored in the RESOURCE_MANAGER property on the root window.

application-specific directory
Any application- or machine-specific resources can be stored in the class resource files located in the /usr/lib/X11/app-defaults directory.

$XENVIRONMENT
Any user- and machine-specific resources may be specified by setting the $XENVIRONMENT environment variable to the name of a resource file to be loaded by all applications. If this variable is not defined, the X Toolkit looks for a file named .Xdefaults-*hostname*, where *hostname* is the name of the host where the application is executing.

-xrm *resourcestring*
Applications that use the X Toolkit can have resources specified from the command line. The *resourcestring* is a single resource name and value as shown above. Note that if the string contains characters interpreted by the shell (e.g., asterisk), they must be quoted. Any number of **-xrm** arguments may be given on the command line.

Program resources are organized into groups called "classes," so that collections of individual "instance" resources can be set all at once. By convention, the instance name of a resource begins with a lowercase letter and class name with an upper case letter. Multiple word resources are concatenated with the first letter of the succeeding words capitalized. Applications written with the X Toolkit will have at least the following resources:

background (class Background)
This resource specifies the color to use for the window background.

borderWidth (class **BorderWidth**)
This resource specifies the width in pixels of the window border.

borderColor (class **BorderColor**)
This resource specifies the color to use for the window border.

Most X Toolkit applications also have the resource **foreground** (class **Foreground**), specifying the color to use for text and graphics within the window.

By combining class and instance specifications, application preferences can be set quickly and easily. Users of color displays will frequently want to set Background and Foreground classes to particular defaults. Specific color instances such as text cursors can then be overridden without having to define all of the related resources.

When a named resource is unavailable (for example, a color named chartreuse or a font named teeneyweeney), normally no error message will be printed; whether or not useful results ensue is dependent on the particular application. If you wish to see error messages (for example, if an application is failing for an unknown reason), you may specify the value "on" for the resource named "StringConversionWarnings." If you want such warnings for all applications, specify "*StringConversionWarnings:on" to the resource manager. If you want warnings only for a single application named "zowie", specify "zowie*StringConversionWarnings:on" to the resource manager.

The available colors are found in the file /usr/lib/X11/rgb.txt. See *rgb(1)* for information on creating a new color database.

DIAGNOSTICS

The default error handler uses the Resource Manager to build diagnostic messages when error conditions arise. The default error database is stored in the file XErrorDB in the /usr/lib/X11 directory. If this file is not installed, error messages will tend to be somewhat cryptic.

COPYRIGHT

ORIGIN

MIT Distribution

SEE ALSO

bitmap(1), gwindstop(1), hpterm(1), init(1M), rgb(1), uwm(1), x10tox11(1), x11start(1), xclock(1), xfc(1), xfd(1), xhost(1), xinit(1), xinitcolormap(1), xload(1), xmodmap(1), xrefresh(1), xseethru(1), Xserver(1), xset(1), xsetroot(1), xterm(1), xwcreate(1), xwd(1), xwdestroy(1), xwininfo(1), xwud(1), *Programming With Xlib, Programming With the HP X Widgets and Xt Intrinsics*

Series 300 and 800 Only

NAME

x11start - start the X11 window system

SYNOPSIS

x11start [options]

DESCRIPTION

x11start starts up the X window system by running the X11 window server and selected X11 clients. By default, *hpterm(1)* and *mwm(1)* are run by *x11start*.

x11start is a shell script that first checks the user's home directory for a *.Xdefaults* file. If there is no *.Xdefaults* file there, then the file */usr/lib/X11/sys.Xdefaults* (if it exists) will be used as a source file for *xrdb(1)*, to create a RESOURCE_MANAGER property. This is done by setting the variable $doxrdb, which is executed in the .x11start file, and thus under user control.

x11start will modify the PATH variable as needed to assure that /usr/bin/X11 is in the users PATH variable in front of /usr/bin.

x11start then runs *xinit* using the shell script *.x11start* from the user's home directory as the first argument for *xinit*. If that script does not exist or is not executable, then the script */usr/lib/X11/sys.x11start* is used as the argument for *xinit*. In any case the arguments passed to *x11start* are passed on to *xinit* following the *.x11start* argument.

FILES

/usr/lib/X11/sys.x11start
$HOME/.x11start
/usr/lib/X11/sys.Xdefaults
$HOME/.Xdefaults

ORIGIN

Hewlett-Packard Company

SEE ALSO

X(1), xinit(1), hpterm(1), mwm(1), xrdb(1)

Series 300 and 800 Only

NAME

xclock - analog / digital clock for X

SYNOPSIS

xclock [*toolkitoptions*] [options]

DESCRIPTION

The *xclock* program displays the time in analog or digital form. The time is continuously updated at a frequency which may be specified by the user. This program is nothing more than a wrapper around the Athena Clock widget.

OPTIONS

xclock accepts all of the standard X Toolkit command line options along with the additional options listed below:

-help This option indicates that a brief summary of the allowed options should be printed on the standard error.

-analog This option indicates that a conventional 12 hour clock face with tick marks and hands should be used. This is the default.

-digital This option indicates that a 24 hour digital clock should be used.

-chime This option indicates that the clock should chime once on the half hour and twice on the hour.

-hd *color*
This option specifies the color of the hands on an analog clock. The default is *black*.

-hl *color* This option specifies the color of the edges of the hands on an analog clock, and is only useful on color displays. The default is *black*.

-update *seconds*
This option specifies the frequency in seconds at which *xclock* should update its display. If the clock is obscured and then exposed, it will be updated immediately. A value of less than 30 seconds will enable a second hand on an analog clock. The default is 60 seconds.

-padding *number*
This option specifies the width in pixels of the padding between the window border and clock text or picture. The default is 10 on a digital clock and 8 on an analog clock.

The following standard X Toolkit command line arguments are commonly used with *xclock:*

-bg *color*
This option specifies the color to use for the background of the window. The default is *white*.

-bd *color*
This option specifies the color to use for the border of the window. The default is *black*.

-bw *number*
This option specifies the width in pixels of the border surrounding the window.

-fg *color* This option specifies the color to use for displaying text. The default is *black*.

-fn *font* This option specifies the font to be used for displaying normal text. The default is *6x10*.

-rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

-geometry *geometry*
This option specifies the preferred size and position of the clock window.

-display *host:display*
This option specifies the X server to contact.

-xrm *resourcestring*
This option specifies a resource string to be used.

X DEFAULTS

This program uses the *Clock* widget in the X Toolkit. It understands all of the core resource

names and classes as well as:

width (class **Width**)
Specifies the width of the clock. The default for analog clocks is 164 pixels; the default for digital clocks is whatever is needed to hold the clock when displayed in the chosen font.

height (class **Height**)
Specifies the height of the clock. The default for analog clocks is 164 pixels; the default for digital clocks is whatever is needed to hold the clock when displayed in the chosen font.

update (class **Interval**)
Specifies the frequency in seconds at which the time should be redisplayed.

foreground (class **Foreground**)
Specifies the color for the tic marks. The default is *black* since the core default for background is *white*.

hands (class **Foreground**)
Specifies the color of the insides of the clock's hands.

highlight (class **Foreground**)
Specifies the color used to highlight the clock's hands.

analog (class **Boolean**)
Specifies whether or not an analog clock should be used instead of a digital one. The default is True.

chime (class **Boolean**)
Specifies whether or not a bell should be rung on the hour and half hour.

padding (class **Margin**)
Specifies the amount of internal padding in pixels to be used. The default is 8.

font (class **Font**)
Specifies the font to be used for the digital clock. Note that variable width fonts currently will not always display correctly.

reverseVideo (class **ReverseVideo**)
Specifies that the foreground and background colors should be reversed.

SEE ALSO

X(1), xrdb(1), time(3C)

ENVIRONMENT

DISPLAY - the default host and display number.

BUGS

xclock believes the system clock.

When in digital mode, the string should be centered automatically.

Border color has to be explicitly specified when reverse video is used.

COPYRIGHT

ORIGIN

Tony Della Fera (MIT-Athena, DEC)
Dave Mankins (MIT-Athena, BBN)
Ed Moy (UC Berkeley)

NAME

xfd - font displayer for X

SYNOPSIS

xfd [options]

OPTIONS

-display *display*
Specifies the display to use.

-geometry *geometry*
Specifies an initial window geometry.

-bw *number*
Allows you to specify the width of the window border in pixels.

-rv The foreground and background colors will be switched. The default colors are black on white.

-fw Overrides a previous choice of reverse video. The foreground and background colors will not be switched.

-fg *color* On color displays, determines the foreground color (the color of the text).

-bg *color*
On color displays, determines the background color.

-bd *color*
On color displays, determines the color of the border.

-fn *fontname*
Specifies the font to be displayed.

-bf *fontname*
Specifies the font to be used for the messages at the bottom of the window.

-tl *title* Specifies that the title of the displayed window should be *title*.

-in *iconname*
Specifies that the name of the icon should be *iconname*.

-icon *filename*
Specifies that the bitmap in file *filename* should be used for the icon.

-verbose Specifies that extra information about the font should be displayed.

-gray Specifies that a gray background should be used.

-start *charnum*
Specifies that character number *charnum* should be the first character displayed.

DESCRIPTION

xfd creates a window in which the characters in the named font are displayed. The characters are shown in increasing order from left to right, top to bottom. The first character displayed at the top left will be character number 0 unless the -start option has been supplied in which case the character with the number given in the -start option will be used.

The characters are displayed in a grid of boxes, each large enough to hold any single character in the font. If the -gray option has been supplied, the characters will be displayed using XDrawImageString using the foreground and background colors on a gray background. This permits determining exactly how XDrawImageString will draw any given character. If -gray has not been supplied, the characters will simply be drawn using the foreground color on the background color.

All the characters in the font may not fit in the window at once. To see additional characters, click the right mouse button on the window. This will cause the next window full of characters to be displayed. Clicking the left mouse button on the window will cause the previous window full of characters to be displayed. *xfd* will beep if an attempt is made to go back past the 0th character.

Note that if the font is a 8 bit font, the characters 256-511 (100-1ff in hexidecimal), 512-767 (200-2ff in hexidecimal), ... will display exactly the same as the characters 0-255 (00-ff in hexidecimal). *xfd* by default creates a window big enough to display 16 rows of 16 columns (totally 256 characters).

Clicking the middle button on a character will cause that character's number to be displayed in both decimal and hexidecimal at the bottom of the window. If verbose mode is selected, additional information about that particular character will be displayed as well. The displayed information includes the width of the character, its left bearing, right bearing, ascent, and its descent. If verbose mode is selected, typing '<' or '>' into the window will display the minimum or maximum values respectively taken on by each of these fields over the entire font.

The font name is interpreted by the X server. To obtain a list of all the fonts available, see the */usr/lib/X11/fonts* directory and its subdirectories.

The window stays around until the xfd process is killed or one of 'q', 'Q', ' ', or ctrl-c is typed into the *xfd* window.

X DEFAULTS

xfd uses the following X resources:

BorderWidth	Set the border width of the window.
BorderColor	Set the border color of the window.
ReverseVideo	If "on", reverse the definition of foreground and background color.
Foreground	Set the foreground color.
Background	Set the background color.
BodyFont	Set the font to be used in the body of the window. (I.e., for messages, etc.) This is not the font that *xfd* displays, just the font it uses to display information about the font being displayed.
IconName	Set the name of the icon.
IconBitmap	Set the file we should look in to get the bitmap for the icon.
Title	Set the title to be used.

ENVIRONMENT

DISPLAY - the default host and display number.

SEE ALSO

X(1), xlsfonts(1), xrdb(1)

BUGS

It should display the name of the font somewhere.

Character information displayed in verbose mode is sometimes clipped to the window boundary, hiding it from view.

It should be rewritten to use the X toolkit.

It should skip over pages full of non-existent characters.

COPYRIGHT

ORIGIN

Mark Lillibridge, MIT Project Athena

Series 300 and 800 Only

NAME

xhost - server access control program for X

SYNOPSIS

xhost [[+-]*hostname* ...]

DESCRIPTION

The *xhost* program is used to add and delete hosts to the list of machines that are allowed to make connections to the X server. This provides a rudimentary form of privacy control and security. It is only sufficient for a workstation (single user) environment, although it does limit the worst abuses. Environments which require more sophisticated measures should use the hooks in the protocol for passing authentication data to the server.

The server initially allows network connections only from programs running on the same machine or from machines listed in the file */etc/X*.hosts* (where * is the display number of the server). The *xhost* program is usually run either from a startup file or interactively to give access to other users.

Hostnames that are followed by two colons (::) are used in checking DECnet connections; all other hostnames are used for TCP/IP connections.

OPTIONS

xhost accepts the command line options described below. For security, the options that effect access control may only be run from the same machine as the server.

[+]*hostname*
: The given *hostname* (the plus sign is optional) is added to the list of machines that are allowed to connect to the X server.

-*hostname*
: The given *hostname* is removed from the list of machines that are allowed to connect to the server. Existing connections are not broken, but new connection attempts will be denied. Note that the current machine is allowed to be removed; however, further connections (including attempts to add it back) will not be permitted. Resetting the server (thereby breaking all connections) is the only way to allow local connections again.

\+
: Access is granted to everyone, even if they aren't on the list of allowed hosts (i.e. access control is turned off).

\-
: Access is restricted to only those machines on the list of allowed hosts (i.e. access control is turned on).

nothing
: If no command line arguments are given, the list of hosts that are allowed to connect is printed on the standard output along with a message indicating whether or not access control is currently enabled. This is the only option that may be used from machines other than the one on which the server is running.

FILES

/etc/X*.hosts

SEE ALSO

X(1), Xserver(1)

ENVIRONMENT

DISPLAY
: to get the host and display to use.

NOTES

You can't specify a display on the command line because **-display** is a valid command line argument (indicating that you want to remove the machine named *"display"* from the access list).

COPYRIGHT

ORIGIN

MIT Distribution

Series 300 and 800 Only

NAME

xinit - X Window System initializer

SYNOPSIS

xinit [[client] options] [-- [server] [display] options]

DESCRIPTION

The *xinit* program is used to start the X Window System server and a first client program (usually a terminal emulator) on systems that do not start X directly from */etc/init* or in environments that use multiple window systems. When this first client exits, *xinit* will kill the X server and then terminate.

Unless otherwise specified on the command line, *xinit* sets $DISPLAY to *hostname*:0.0, and exports it. *hostname* is the name of the system invoking *xinit* as returned by gethostname(2).

If no specific client program is given on the command line, *xinit* will look for a file in the user's home directory called *.xinitrc* to run as a shell script to start up client programs. If no such file exists, *xinit* will use the following as a default:

```
xterm -geometry +1+1 -n login
```

If no specific server program is given on the command line, *xinit* will look for a file in the user's home directory called *.xserverrc* to run as a shell script to start up the server. If no such file exists, *xinit* will use the following as a default:

```
X :0
```

Note that this assumes that there is a program named *X* in the current search path.

An important point is that programs which are run by *.xinitrc* and by *.xserverrc* should be run in the background if they do not exit right away, so that they don't prevent other programs from starting up. However, the last long-lived program started (usually a window manager or terminal emulator) should be left in the foreground so that the script won't exit (which indicates that the user is done and that *xinit* should exit).

An alternate client and/or server may be specified on the command line. The desired client program and its arguments should be given as the first command line arguments to *xinit*. To specify a particular server command line, append a double dash (--) to the *xinit* command line (after any client and arguments) followed by the desired server comand.

Both the client program name and the server program name must begin with a slash (/) or a period (.). Otherwise, they are treated as an arguments to be appended to their respective startup lines. This makes it possible to add arguments (for example, foreground and background colors) without having to retype the whole command line.

If an explicit server name is not given and the first argument following the double dash (--) is a colon followed by a digit, *xinit* will use that number as the display number instead of zero. It will be incorporated into the DISPLAY variable. All remaining arguments are appended to the server command line.

EXAMPLES

Below are several examples of how command line arguments in *xinit* are used.

xinit This will start up a server named *X* if *.xserverrc* doesn't exist, and run the user's *.xinitrc*, if it exists, or else start an *xterm*.

xinit -- /usr/bin/X11/Xqdss :1
This is how one could start a specific type of server on an alternate display.

xinit -geometry =80x65+10+10 -fn 8x13 -j -fg white -bg navy
This will start up a server named *X* or *.xserverrc*, and will append the given arguments to the default *xterm* command. It will ignore *.xinitrc*.

xinit -e widgets -- ./Xsun -l -c
This will use the command *./Xsun -l -c* to start the server and will append the arguments

-e widgets to the default *xterm* command.

xinit remsh fasthost cpupig -display ws:1 -- :1 -a 2 -t 5
This will start a server named *X* on display 1 with the arguments *-a 2 -t 5*. It will then start a remote shell on the machine **fasthost** in which it will run the command *cpupig*, telling it to display back on the local workstation.

Below is a sample *.xinitrc* that starts a clock, several terminals, and leaves the window manager running as the "last" application. Assuming that the window manager has been configured properly, the user then chooses the "Exit" menu item to shut down X.

```
xrdb -load $HOME/.Xres
xsetroot -solid gray &
xclock -geometry 50x50-0+0 -bw 0 &
xload -geometry 50x50-50+0 -bw 0 &
hpterm -geometry 80x24+0+0 &
hpterm -geometry 80x24+0-0 &
mwm
```

Sites that want to create a common startup environment could simply create a default *.xinitrc* that references a site-wide startup file:

```
#!/bin/sh
. /usr/local/lib/site.xinitrc
```

Another approach is to write a script that starts *xinit* with a specific shell script. Such scripts are usually named *x11*, *xstart*, or *startx* and are a convenient way to provide a simple interface for novice users:

```
#!/bin/sh
xinit /usr/local/bin/startx -- /usr/bin/X11/X :1
```

Hewlett Packard supplies /usr/bin/x11start and /usr/lib/X11/sys.x11start for this purpose.

ENVIRONMENT VARIABLES

DISPLAY
If not already set, gets set to hostname:0.0 or hostname:number.0, where number is specified on the command line to the server (as in xinit -- :1). If *DISPLAY* is already set, the display number is passed to the server. *DISPLAY* specifies the name of the display to which clients may connect.

XINITRC
This variable specifies an init file containing shell commands to start up the initial windows. By default, *.xinitrc* in the home directory will be used.

XSERVERC
This variable specifies an init file containing shell commands to start up the server. By default, *.xserverrc* in the home directory will be used.

SEE ALSO

X(1), Xserver(1), xterm(1), xrdb(1), x11start(1)

COPYRIGHT

ORIGIN

MIT Distribution

Series 300 and 800 Only

NAME

xinitcolormap - initialize the X colormap

SYNOPSIS

xinitcolormap [options]

DESCRIPTION

This program is used to initialize the X colormap. Specific X colormap entries (pixel values) are made to correspond to specified colors. An initialized colormap is required by applications that assume a predefined colormap (e.g., many applications that use *Starbase* graphics).

xinitcolormap reads a colormap file to determine the allocation of colors in the X colormap. The name of the colormap file is determined by using (in the following order) the command line option [*-f colormapfile*], the *.Colormap* X default value or */usr/lib/X11/xcolormap*. If a colormap file is not found, then the following default colormap specification is assumed.

black (colormap entry 0)
white
red
yellow
green
cyan
blue
magenta (colormap entry 7)

xinitcolormap uses the *XStoreColor* and *XAllocColor libX11.a* calls to initialize the X colormap. The *xinitcolormap* program should be the first X client program run when the *X Window System* is started in order to assure that X colormap entries have the color associations specified in the colormap file. This could be done by running *xinitcolormap* as the first X client program in the *.x11start* file. Once *xinitcolormap* has been run, an X client program can use the initialized colors.

A colormap file is made up of lines of the form:

color

color is a one or two word color name (refer to the names in the file */usr/lib/X11/rgb.txt*), or optionally an initial sharp character followed by a numeric RGB specification (as used by the *libX.a* call *XParseColor*). The line number of a color specification in the colormap file determines the index of the color in the X colormap. Colors in the colormap file, for colormap entry 0 up to the last colormap entry to be initialized, must be specified. There should be no extra (blank or comment) lines in the colormap file. The first two entries in the colormap file must be black and white. Also, a color may be specified more than once in the colormap file.

OPTIONS

-f *colormapfile*
Specifies the file containing the colormap.

-display *display*
Specifies the server to connect to; See *X(1)* for details.

-c *count* If *count* is specified then only the first *count* colors from the colormap file will be used in initializing the X colormap.

-p If the *-p* option is specified then the colormap file will be checked for proper syntax, but the X colormap will not be initialized.

-k[ill] If the *-k*[*ill*] option is specified, then the colormap entries allocated by a previous run of *xinitcolormap* will be deallocated and the colormap will not be re-initialized. All other options will be ignored except **-display** *display*.

NOTES

xinitcolormap will only initialize the default colormap of the root window.

xinitcolormap assumes the first two colors specified are black and white.

xinitcolormap should not be run in the background. The X colormap is fully initialized only when *xinitcolormap* returns.

Running *xinitcolormap* a second time after X is started will deallocate those colors allocated by a previous run and attempt to allocate a new colormap using the new specifications. If other clients have allocated color cells that conflict with the new specifications, *xinitcolormap* will fail and the colormap will remain un-allocated.

The file */etc/newconfig/xcolormap* is a sample colormap file that corresponds to the *Starbase* default 256 entry colormap. The [*-c count*] option can be used to select a subset of the colors in this colormap file for initializing colormaps with up to 256 entries.

xinitcolormap uses XSetCloseDownMode with RetainPermanent to prevent the deallocation of the colormap. This means that *xinitcolormap* no longer spawns a daemon, and the only way for the user to be sure that *xinitcolormap* succeeded is to view the messages (or lack of) produced by *xinitcolormap*. If *x11start* is used, the output should be redirected from *xinitcolormap* to a log file.

xinitcolormap will not work on TrueColor visuals.

FILES

/usr/lib/X11/xcolormap
/usr/lib/X11/rgb.txt
/etc/newconfig/xcolormap
.x11start

Series 300 and 800 Only

NAME

xload - load average display for X

SYNOPSIS

xload [*toolkitoptions*] [options]

DESCRIPTION

The *xload* program displays a periodically updating histogram of the system load average. This program is nothing more than a wrapper around the Athena Load widget.

OPTIONS

xload accepts all of the standard X Toolkit command line options along with the following additional options:

-scale *integer*
This option specifies the minimum number of tick marks in the histogram, where one division represents one load average point. If the load goes above this number, *xload* will create more divisions, but it will never use fewer than this number. The default is 1.

-update *seconds*
This option specifies the frequency in seconds at which *xload* updates its display. If the load average window is uncovered (by moving windows with a window manager or by the *xrefresh* program), the graph will be also be updated. The minimum amount of time allowed between updates is 1 second. The default is 5 seconds.

-hl *color* This option specifies the color of the label and scale lines.

The following standard X Toolkit arguments are commonly used with *xload*:

-bd *color*
This option specifies the border color. The default is *black*.

-bg *color*
This option specifies the background color. The default is *white*.

-bw *pixels*
This option specifies the width in pixels of the border around the window. The default value is 2.

-fg *color* This option specifies the graph color. The default color is *black*.

-fn *fontname*
This option specifies the font to be used in displaying the name of the host whose load is being monitored. The default is *6x10*.

-rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

-geometry *geometry*
This option specifies the preferred size and postion of the window.

-display *display*
This option specifies the X server to contact.

-xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

X DEFAULTS

This program uses the *Load* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width**)
Specifies the width of the load average graph.

height (class **Height**)
Specifies the height of the load average graph.

update (class **Interval**)
Specifies the frequency in seconds at which the load should be redisplayed.

scale (class **Scale**)
Specifies the initial number of ticks on the graph. The default is 1.

minScale (class **Scale**)
Specifies the minimum number of ticks that will be displayed. The default is 1.

foreground (class **Foreground**)
Specifies the color for the graph. The default is *black* since the core default for background is *white*.

highlight (class **Foreground**)
Specifies the color for the text and scale lines. The default is the same as for the **foreground** resource.

label (class **Label**)
Specifies the label to use on the graph. The default is the hostname.

font (class **Font**)
Specifies the font to be used for the label. The default is *fixed*.

reverseVideo (class **ReverseVideo**)
Specifies that the foreground and background should be reversed.

ENVIRONMENT
DISPLAY - the default host and display number.

SEE ALSO
X(1), xrdb(1), mem(4), Athena Load widget

DIAGNOSTICS
Unable to open display or create window. Unable to open /dev/kmem. Unable to query window for dimensions. Various X errors.

BUGS
This program requires the ability to open and read the special system file */dev/kmem*. Sites that do not allow general access to this file should make *xload* belong to the same group as */dev/kmem* and turn on the *set group id* permission flag.

Reading /dev/kmem is inherently non-portable. Therefore, the widget upon which this application is based must be ported to each new operating system.

Border color has to be explicitly specified when reverse video is used.

ORIGIN
K. Shane Hartman (MIT-LCS) and Stuart A. Malone (MIT-LCS);
with features added by Jim Gettys (MIT-Athena), Bob Scheifler (MIT-LCS), and Tony Della Fera (MIT-Athena)

Series 300 and 800 Only

NAME

xlsfonts - server font list displayer for X

SYNOPSIS

xlsfonts [-options ...] [-fn pattern]

DESCRIPTION

Xlsfonts lists the fonts that match the given *pattern*. The wildcard character "*" may be used to match any sequence of characters (including none), and "?" to match any single character. If no pattern is given, "*" is assumed.

The "*" and "?" characters must be quoted to prevent them from being expanded by the shell.

OPTIONS

-display *host:dpy*
This option specifies the X server to contact.

-l This option indicates that a long listing should be generated for each font.

-L This option indicates that a very long listing showing the individual character metrics should be printed.

-m This option indicates that long listings should also print the minimum and maximum bounds of each font.

-C This option indicates that listings should use multiple columns. This is the same as **-n 0**.

-1 This option indicates that listings should use a single column. This is the same as **-n 1**.

-w *width* This option specifies the width in characters that should be used in figuring out how many columns to print. The default is 79.

-n *columns*
This option specifies the number of columns to use in displaying the output. By default, it will attempt to fit as many columns of font names into the number of character specified by **-w** *width*.

SEE ALSO

X(1), Xserver(1), xset(1), xfd(1)

ENVIRONMENT

DISPLAY
to get the default host and display to use.

BUGS

Doing "xlsfonts -l" can tie up your server for a very long time. This is really a bug with single-threaded non-preemptible servers, not with this program.

COPYRIGHT

ORIGIN

Mark Lillibridge, MIT Project Athena

Series 300 and 800 Only

NAME

xmodmap - utility for modifying keymaps in X

SYNOPSIS

xmodmap [-options ...] [filename]

DESCRIPTION

The *xmodmap* program is used to edit or display the keyboard *modifier map* and *keymap table* that are used by client applications to convert event keycodes into keysyms. It is usually run from the user's session startup script to configure the keyboard according to personal tastes.

OPTIONS

The following options may be used with *xmodmap*:

-display *display*
This option specifies the host and display to use.

-help This option indicates that a brief description of the command line arguments should be printed on the standard error. This will be done whenever an unhandled argument is given to *xmodmap*.

-grammar
This option indicates that a help message describing the expression grammar used in files and with -e expressions should be printed on the standard error.

-verbose This option indicates that *xmodmap* should print logging information as it parses its input.

-quiet This option turns off the verbose logging. This is the default.

-n This option indicates that *xmodmap* should not change the mappings, but should display what it would do, like *make(1)* does when given this option.

-e *expression*
This option specifies an expression to be executed. Any number of expressions may be specified from the command line.

-pm This option indicates that the current modifier map should be printed on the standard output. This is the default.

-pk This option indicates that the current keymap table should be printed on the standard output.

-pp This option indicates that the current pointer map should be printed on the standard output.

- A lone dash means that the standard input should be used as the input file.

The *filename* specifies a file containing *xmodmap* expressions to be executed. This file is usually kept in the user's home directory with a name like *.xmodmaprc*.

EXPRESSION GRAMMAR

The *xmodmap* program reads a list of expressions and parses them all before attempting execute any of them. This makes it possible to refer to keysyms that are being redefined in a natural way without having to worry as much about name conflicts.

keycode *NUMBER = KEYSYMNAME ...*
The list of keysyms is assigned to the indicated keycode (which may be specified in decimal, hex or octal) The order of assignment is no modifier, shift, mod1, shift+mod1.

keysym *KEYSYMNAME = KEYSYMNAME ...*
The *KEYSYMNAME* on the left hand side is looked up to find its current keycode and the line is replaced with the appropriate **keycode** expression. Note that if you have the same keysym bound to multiple keys, this might not work.

clear *MODIFIERNAME*
This removes all entries in the modifier map for the given modifier, where valid name are: Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4 and Mod5 (case does not matter in modifier names, although it does matter for all other names). For example, "clear Lock"

will remove all any keys that were bound to the shift lock modifier.

add *MODIFIERNAME = KEYSYMNAME ...*
This adds the given keysyms to the indicated modifier map. The keysym names are evaluated after all input expressions are read to make it easy to write expressions to swap keys (see the EXAMPLES section).

remove *MODIFIERNAME = KEYSYMNAME ...*
This removes the given keysyms from the indicated modifier map. Unlike **add,** the keysym names are evaluated as the line is read in. This allows you to remove keys from a modifier without having to worry about whether or not they have been reassigned.

pointer = default
This sets the pointer map back to its default settings (button 1 generates a code of 1, button 2 generates a 2, etc.).

pointer = *NUMBER ...*
This sets to pointer map to contain the indicated button codes. The list always starts with the first physical button.

Lines that begin with an exclamation point (!) are taken as comments.

If you want to change the binding of a modifier key, you must also remove it from the appropriate modifier map.

EXAMPLES

Many pointers are designed such the first button is pressed using the index finger of the right hand. People who are left-handed frequently find that it is more comfortable to reverse the button codes that get generated so that the primary button is pressed using the index finger of the left hand. This could be done on a 3 button pointer as follows:

```
% xmodmap -e "pointer = 3 2 1"
```

Many editor applications support the notion of Meta keys (similar to Control keys except that Meta is held down instead of Control). However, some servers do not have a Meta keysym in the default keymap table, so one needs to be added by hand. The following command will attach Meta to the Select key. It also takes advantage of the fact that applications that need a Meta key simply need to get the keycode and don't require the keysym to be in the first column of the keymap table. This means that applications that are looking for a Multi_key (including the default modifier map) won't notice any change.

```
% xmodmap -e 'keysym Select = Select Meta_L'
```

One of the more simple, yet convenient, uses of *xmodmap* is to set the keyboard's "rubout" key to generate an alternate keysym. This frequently involves exchanging Backspace with Delete to be more comfortable to the user. If the *ttyModes* resource in *xterm* is set as well, all terminal emulator windows will use the same key for erasing characters:

```
% xmodmap -e "keysym BackSpace = Delete"
% echo "XTerm*ttyModes:  erase ^?" | xrdb -merge
```

Some keyboards do not automatically generate less than and greater than characters when the comma and period keys are shifted. This can be remedied with *xmodmap* by resetting the bindings for the comma and period with the following scripts:

```
!
! make shift-, be < and shift-. be >
!
keysym comma = comma less
keysym period = period greater
```

One of the more irritating differences between keyboards is the location of the Control and Shift

Lock keys. A common use of *xmodmap* is to swap these two keys as follows:

```
!
! Swap Caps_Lock and Control_L
!
remove Lock = Caps_Lock
remove Control = Control_L
keysym Control_L = Caps_Lock
keysym Caps_Lock = Control_L
add Lock = Caps_Lock
add Control = Control_L
```

The *keycode* command is useful for assigning the same keysym to multiple keycodes. Although unportable, it also makes it possible to write scripts that can reset the keyboard to a known state. The following script sets the backspace key to generate Delete (as shown above), flushes all existing caps lock bindings, makes the CapsLock key be a control key, make F5 generate Escape, and makes Break/Reset be a shift lock.

```
!
! On the HP, the following keycodes have key caps as listed:
!
!   101 Backspace
!    55 Caps
!    14 Ctrl
!    15 Break/Reset
!    86 Stop
!    89 F5
!
keycode 101 = Delete
keycode 55 = Control_R
clear Lock
add Control = Control_R
keycode 89 = Escape
keycode 15 = Caps_Lock
add Lock = Caps_Lock
```

ENVIRONMENT

DISPLAY

to get default host and display number.

SEE ALSO

X(1)

NOTES

Every time a **keycode** expression is evaluated, the server generates a *MappingNotify* event on every client. This can cause some thrashing. All of the changes should be batched together and done at once. Clients that receive keyboard input and ignore *MappingNotify* events will not notice any changes made to keyboard mappings.

Xmodmap should generate "add" and "remove" expressions automatically whenever a keycode that is already bound to a modifier is changed.

There should be a way to have the *remove* expression accept keycodes as well as keysyms for those times when you really mess up your mappings.

ORIGIN

MIT Distribution

Series 300 and 800 Only

NAME

xpr - print an X window dump

SYNOPSIS

xpr [options] [*filename*]

DESCRIPTION

xpr takes as input a window dump file produced by *xwd(1)* and formats it for output on the HP LaserJet (or other PCL printers), HP PaintJet, LN03, LA100, PostScript printers, or IBM PP3812 page printer. If no *filename* argument is given, the standard input is used. By default, *xpr* prints the largest possible representation of the window on the output page. Options allow the user to add headers and trailers, specify margins, adjust the scale and orientation, and append multiple window dumps to a single output file. Output is to standard output unless **-output** is specified.

Command Options

-scale *scale*
Affects the size of the window on the page. The HP, LN03 and PostScript printers are able to translate each bit in a window pixel map into a grid of a specified size. For example each bit might translate into a 3x3 grid. This would be specified by **-scale** *3*. By default a window is printed with the largest scale that will fit onto the page for the specified orientation.

-density *dpi*
Indicates what dot-per-inch density should be used by the printer.

-height *inches*
Specifies the maximum height of the page.

-width *inches*
Specifies the maximum width of the page.

-left *inches*
Specifies the left margin in inches. Fractions are allowed. By default the window is centered in the page.

-top *inches*
Specifies the top margin for the picture in inches. Fractions are allowed.

-header *string*
Specifies a header string to be printed above the window.

-trailer *string*
Specifies a trailer string to be printed below the window.

-landscape
Forces the window to printed in landscape mode. By default a window is printed such that its longest side follows the long side of the paper.

-portrait
Forces the window to be printed in portrait mode. By default a window is printed such that its longest side follows the long side of the paper.

-rv Forces the window to be printed in reverse video.

-compact
Uses simple run-length encoding for compact representation of windows with lots of white pixels.

-output *filename*
Specifies an output file name. If this option is not specified, standard output is used.

-append *filename*
Specifies a filename previously produced by *xpr* to which the window is to be appended.

-noff When specified in conjunction with **-append**, the window will appear on the same page as the previous window.

-split *n* This option allows the user to split a window onto several pages. This might be necessary for very large windows that would otherwise cause the printer to overload and print the page in an obscure manner.

-device *dev*

Specifies the device on which the file will be printed. Currently *xpr* understands the following *dev*s:

ljet	HP LaserJet series and other monochrome PCL devices such as ThinkJet, QuietJet, RuggedWriter, HP2560 series, and HP2930 series printers
pjet	HP PaintJet (color mode)
pjetxl	HP HP PaintJet XL Color Graphics Printer (color mode).
ln03	DEC LN03
la100	DEC LA100
ps	PostScript printers
pp	IBM PP3812

The default device is *ljet*. **-device lw** (LaserWriter) is equivalent to -device ps and is provided only for backwards compatibility.

-cutoff *level*

Changes the intensity level where colors are mapped to either black or white for monochrome output on a LaserJet printer. The *level* is expressed as percentage of full brightness. Fractions are allowed.

-noposition

This option causes header, trailer, and image positioning command generation to be bypassed for LaserJet, PaintJet and PaintJet XL printers.

-gamma *correction*

This changes the intensity of the colors printed by PaintJet XL printer. The *correction* is a floating point value in the range 0.00 to 3.00. Consult the operator's manual to determine the correct value for the specific printer.

-render *algorithm*

This allows PaintJet XL printer to render the image with the best quality versus performance tradeoff. Consult the operator's manual to determine which *algorithm*s are available.

-slide This option allows overhead transparencies to be printed using the PaintJet and PaintJet XL printers.

LIMITATIONS

The current version of *xpr* can generally print out on the LN03 most X windows that are not larger than two-thirds of the screen. For example, it will be able to print out a large Emacs window, but it will usually fail when trying to print out the entire screen. The LN03 has memory limitations that can cause it to print incorrectly very large or complex windows. The two most common errors encountered are "band too complex" and "page memory exceeded." In the first case, a window may have a particular six pixel row that contains too many changes (from black to white to black). This will cause the printer to drop part of the line and possibly parts of the rest of the page. The printer will flash the number '1' on its front panel when this problem occurs. A possible solution to this problem is to increase the scale of the picture, or to split the picture onto two or more pages. The second problem, "page memory exceeded," will occur if the picture contains too much black, or if the picture contains complex half-tones such as the background color of a display. When this problem occurs the printer will automatically split the picture into two or more pages. It may flash the number '5' on its front panel. There is no easy solution to this problem. It will probably be necessary to either cut and paste, or rework to application to produce a less complex picture.

xpr provides some support for the LA100. However, there are several limitations on its use: The picture will always be printed in portrait mode, there is no scaling and the aspect ratio will be slightly off.

Support for PostScript output currently cannot handle the **-append**, **-noff** or **-split** options.

The **-compact** option is *only* supported for PostScript output. It compresses white space but not black space, so it is not useful for reverse-video windows.

HP PRINTERS

If no **-density** is specified on the command line 300 dots per inch will be assumed for *ljet* and 90 dots per inch for *pjet*. Allowable *density* values for a LaserJet printer are 300, 150, 100, and 75 dots per inch. Consult the operator's manual to determine densities supported by other printers.

If no **-scale** is specified the image will be expanded to fit the printable page area.

The default printable page area is 8x10.5 inches. Other paper sizes can be accomodated using the **-height** and **-width** options.

Note that a 1024x768 image fits the default printable area when processed at 100 dpi with scale=1, the same image can also be printed using 300 dpi with scale=3 but will require considerably more data be transfered to the printer.

xpr may be tailored for use with monochrome PCL printers other than the LaserJet. To print on a ThinkJet (HP2225A) *xpr* could be invoked as:

xpr -density 96 -width 6.667 *filename*

or for black-and-white output to a PaintJet:

xpr -density 180 *filename*

The monochrome intensity of a pixel is computed as 0.30*R + 0.59*G + 0.11*B. If a pixel's computed intensity is less than the **-cutoff** level it will print as white. This maps light-on-dark display images to black-on-white hardcopy. The default cutoff intensity is 50% of full brightness. Example: specifying **-cutoff 87.5** moves the white/black intensity point to 87.5% of full brightness.

A LaserJet printer must be configured with sufficient memory to handle the image. For a full page at 300 dots per inch approximately 2MB of printer memory is required.

Color images are produced on the PaintJet at 90 dots per inch. The PaintJet is limited to sixteen colors from its 330 color palette on each horizontal print line. *xpr* will issue a warning message if more than sixteen colors are encountered on a line. *xpr* will program the PaintJet for the first sixteen colors encountered on each line and use the nearest matching programmed value for other colors present on the line.

Specifying the **-rv**, reverse video, option for the PaintJet will cause black and white to be interchanged on the output image. No other colors are changed.

Multiplane images must be recorded by *xwd* in *ZPixmap* format. Single plane (monochrome) images may be in either *XYPixmap* or *ZPixmap* format.

Some PCL printers do not recognize image positioning commands. Output for these printers will not be centered on the page and header and trailer strings may not appear where expected.

The **-gamma** and **-render** options are supported only on the PaintJet XL printers.

The **-slide** option is not supported for LaserJet printers.

The **-split** option is not supported for HP printers.

ORIGIN

MIT Distribution

SEE ALSO

xwd(1), xdpr(1), xwud(1), X(1)

NAME

xrdb - X server resource database utility

SYNOPSIS

xrdb [-option ...] [*filename*]

DESCRIPTION

Xrdb is used to get or set the contents of the RESOURCE_MANAGER property on the root window of screen 0. You would normally run this program from your X startup file.

The resource manager (used by the Xlib routine *XGetDefault(3X)* and the X Toolkit) uses the RESOURCE_MANAGER property to get user preferences about color, fonts, and so on for applications. Having this information in the server (where it is available to all clients) instead of on disk, solves the problem in previous versions of X that required you to maintain *defaults* files on every machine that you might use. It also allows for dynamic changing of defaults without editting files.

For compatibility, if there is no RESOURCE_MANAGER property defined (either because xrdb was not run or if the property was removed), the resource manager will look for a file called *.Xdefaults* in your home directory.

Lines that begin with an exclamation mark (!) or (#) are ignored and may be used as comments. However this behavior is dependent on the preprocessor used. *Cpp* for instance, does not use '#' as a comment.

The *filename* (or the standard input if - or no input file is given) is optionally passed through the C preprocessor with the following symbols defined, based on the capabilities of the server being used:

HOST=hostname
the hostname portion of the display to which you are connected.

WIDTH=num
the width of the screen in pixels.

HEIGHT=num
the height of the screen in pixels.

X_RESOLUTION=num
the x resolution of the screen in pixels per meter.

Y_RESOLUTION=num
the y resolution of the screen in pixels per meter.

PLANES=num
the number of bit planes for the default visual.

BITS_PER_RGB=num
the number of significant bits in an RGB color specification. This is the log base 2 of the number of distinct shades of each primary that the hardware can generate. Note that it is not related to the number of planes, which the log base 2 of the size of the colormap.

CLASS=visualclass
one of StaticGray, GrayScale, StaticColor, PseudoColor, TrueColor, DirectColor.

COLOR only defined if the default visual's type is one of the color options.

OPTIONS

xrdb program accepts the following options:

-help This option (or any unsupported option) will cause a brief description of the allowable options and parameters to be printed.

-display *display*
This option specifies the X server to be used; see *X(1)*.

-cpp *filename*
This option specifies the pathname of the C preprocessor program to be used. Although *xrdb* was designed to use CPP, any program that acts as a filter and accepts the -D, -I,

and -U options may be used.

-nocpp This option indicates that *xrdb* should not run the input file through a preprocessor before loading it into the RESOURCE_MANAGER property.

-symbols
This option indicates that the symbols that are defined for the preprocessor should be printed onto the standard output. It can be used in conjunction with **-query,** but not with the options that change the RESOURCE_MANAGER property.

-query This option indicates that the current contents of the RESOURCE_MANAGER property should be printed onto the standard output. Note that since preprocessor commands in the input resource file are part of the input file, not part of the property, they won't appear in the output from this option. The **-edit** option can be used to merge the contents of the property back into the input resource file without damaging preprocessor commands.

-load This option indicates that the input should be loaded as the new value of the RESOURCE_MANAGER property, replacing whatever what there (i.e. the old contents are removed). This is the default action.

-merge This option indicates that the input should be merged with, instead of replacing, the current contents of the RESOURCE_MANAGER property. Since *xrdb* can read the standard input, this option can be used to the change the contents of the RESOURCE_MANAGER property directly from a terminal or from a shell script.

-remove This option indicates that the RESOURCE_MANAGER property should be removed from its window.

-edit *filename*
This option indicates that the contents of the RESOURCE_MANAGER property should be edited into the given file, replacing any values already listed there. This allows you to put changes that you have made to your defaults back into your resource file, preserving any comments or preprocessor lines.

-backup *string*
This option specifies a suffix to be appended to the filename used with **-edit** to generate a backup file.

-D*name*[=*value*]
This option is passed through to the preprocessor and is used to define symbols for use with conditionals such as *#ifdef.*

-U*name* This option is passed through to the preprocessor and is used to remove any definitions of this symbol.

-I*directory*
This option is passed through to the preprocessor and is used to specify a directory to search for files that are referenced with *#include.*

FILES

Generalizes ~/*.Xdefaults* files.

ENVIRONMENT

DISPLAY
to figure out which display to use.

NOTES

The default for no arguments should be to query, not to overwrite, so that it is consistent with other programs.

COPYRIGHT

ORIGIN

MIT Distribution

SEE ALSO

X(1), XGetDefault(3X), Xlib Resource Manager documentation

Series 300 and 800 Only

NAME

xrefresh - refresh all or part of an X screen

SYNOPSIS

xrefresh [options]

DESCRIPTION

xrefresh is a simple X program that causes all or part of your screen to be repainted. *xrefresh* maps a window on top of the desired area of the screen and then immediately unmaps it, causing refresh events to be sent to all applications. By default, a window with no background is used, causing all applications to repaint "smoothly." However, the various options can be used to indicate that a solid background (of any color) or the root window background should be used instead.

OPTIONS

-white Use a white background. The screen just appears to flash quickly, and then repaint.

-black Use a black background (in effect, turning off all of the electron guns to the tube). This can be somewhat disorienting as everything goes black for a moment.

-solid *color*
Use a solid background of the specified color.

-root Use the root window background.

-none This is the default. All of the windows simply repaint.

-geometry *geometry*
Specifies the portion of the screen to be repainted; see *X(1)*.

-display *display*
This argument allows you to specify the server and screen to refresh; see *X(1)*.

X DEFAULTS

The *xrefresh* program uses the routine *XGetDefault(3X)* to read defaults, so its resource names are all capitalized.

Black Uses black for the window background. Default is False.

White Uses white for the window background. Default is False.

Solid Uses the given color for the window background.

None Maps an invisible window to the screen. Default is True.

Root Uses the root window background for the window background. Default is False.

Geometry
Determines the area to refresh.

ENVIRONMENT

DISPLAY
- To get default host and display number.

NOTES

It should have just one default type for the background.

COPYRIGHT

ORIGIN

MIT Distribution

SEE ALSO

X(1)

Series 300 and 800 Only

NAME

xseethru - X Window System, opens a transparent window into the HP 98550A, HP 98720A or HP 98730A Graphics Display Systems overlay plane.

SYNOPSIS

xseethru [options]

DESCRIPTION

xseethru opens a transparent window into the HP 98550A, HP 98720A or HP 98730A Graphics Display System overlay plane. It's used typically to view Starbase graphics in the image planes while running X in the overlay planes.

OPTIONS

-geometry *geometry*

The transparent window is created with the specified size according to the geometry specification. See *X(1)* for details.

-display *display*

Specified the sever to use; see *X(1)* for details.

ENVIRONMENT

DISPLAY

To get the default host and display number.

HARDWARE DEPENDENCIES

Only one display: 0, is supported on the HP 9000 Series 300.
xseethru is only useful on the HP 98550A, HP 98720A or HP 98730A Graphics Display System.

ORIGIN

Hewlett-Packard Company

SEE ALSO

X(1)

Series 300 and 800 Only

NAME

X - X Window System server

SYNOPSIS

X *:displaynumber* [-option] *ttyname*

DESCRIPTION

X is the window system server. It is started from the *xinit(1)* program, which is called by *x11start*. The *displaynumber* argument is used by clients in their DISPLAY environment variables to indicate which server to contact (large machines may have several displays attached). This number can be any number, but there can't be more than 4 of them. If no number is specified 0 is used. This number is also used in determining the names of various startup files. The *ttyname* argument is passed in by *init* and isn't used.

The executable that is invoked when *X* is run is actually one of a collection of programs that depend on the hardware that is installed on the machine. Any additional features are described in the documentation for that server.

The Hewlett-Packard server has support for the following protocols:

TCP/IP

The server listens on port htons(6000+N), where N is the display number.

Local IPC Mechanism

The file name for the socket is /usr/spool/sockets/X11/* where "*" is the display number.

When the server starts up, it takes over the display. If you are running on a workstation whose console is the display, you cannot log into the console while the server is running.

OPTIONS

The following options can be given on the command line to any X server, usually when it is started by *init(1M)*.

-a *number*

sets pointer acceleration (i.e. the ratio of how much is reported to how much the user actually moved the pointer).

-c turns off key-click.

c *volume*

sets key-click volume (allowable range: 0-100). Default is 50.

-f *volume*

sets feep (bell) volume (allowable range: 0-100). Default is 50.

-logo turns on the X Window System logo display in the screen-saver. There is currently no way to change this from a client. Default is -logo. This must be used in conjuction with -v.

nologo turns off the X Window System logo display in the screen-saver. There is currently no way to change this from a client.

-p *minutes*

sets screen-saver pattern cycle time in minutes. Default is 10 minutes.

-r turns off auto-repeat.

r turns on auto-repeat.

-s *minutes*

sets screen-saver timeout time in minutes. Default is 10 minutes.

-t *numbers*

sets pointer acceleration threshold in pixels (i.e. after how many pixels pointer acceleration should take effect).

-to *seconds*

sets connection timeout in seconds. Default is 60 seconds.

v sets video-on screen-saver preference.

-v sets video-off screen-saver preference

-co *filename*
sets name of RGB color database

-help prints a usage message

-fp *fontPath*
sets the search path for fonts

-fc *cursorFont*
sets default cursor font

-fn *font* sets the default font

RUNNING FROM INIT

To run X from *init*, it is necessary to modify */etc/inittab* and */etc/gettydefs*. Detailed information on these files may be obtained from the *inittab*(*4*) and *gettydefs*(*4*) man pages.

To run X from *init* on display 0, with a login *xterm* running on */dev/ttypf*, in init state 3, the following line must be added to */etc/inittab*:

X0:3:respawn:env PATH=/bin:/usr/bin/X11:/usr/bin xinit -L ttyqf -- :0

To run X with a login *hpterm*, the following should be used instead:

X0:3:respawn:env PATH=/bin:/usr/bin/X11:/usr/bin xinit hpterm = +1+1 -n login -L ttyqf -- :0

In addition, the following line must be added to */etc/gettydefs* (this should be a single line):

Xwindow# B9600 HUPCL PARENB CS7 # B9600 SANE PARENB CS7 ISTRIP IXANY TAB3 #X login: #Xwindow

There should not be a getty running against the display for states in which X is run from *xinit*.

SECURITY

X uses an access control list for deciding whether or not to accept a connection from a given client. This list initially consists of the machine on which the server is running, and any hosts listed in the file */etc/X*.hosts* (where * is the display number). This file should contain one line per host name, with no white space.

The user can manipulate a dynamic form of this list in the server using the *xhost*(*1*) program from the same machine as the server.

Unlike some window systems, *X* does not have any notion of window operation permissions or place any restrictions on what a client can do; if a program can connect to a display, it has full run of the screen.

SIGNALS

X will catch the SIGHUP signal sent by *init*(*1M*) after the initial process (usually the login terminal window) started on the display terminates. This signal causes all connections to be closed (thereby "disowning" the terminal), all resources to be freed, and all defaults restored.

DIAGNOSTICS

Too numerous to list them all. If run from *init*(*1M*), errors are logged in the file */usr/adm/X*msgs*,

FILES

/etc/inittab	Script for the init process
/etc/gettydefs	Speed and terminal settings used by getty
/etc/X*.hosts	Initial access control list
/usr/lib/X11/fonts	Font directory
/usr/lib/X11/rgb.txt	Color database
/usr/lib/X11/rgb.pag	Color database

/usr/lib/X11/rgb.dir	Color database
/usr/spool/sockets/X11/*	IPC mechanism socket
/usr/adm/X*msgs	Error log file
/usr/lib/X11/X*devices	Input devices used by the server
/usr/lib/X11/X*screens	Screens used by the server
/usr/lib/X11/X*pointerkeys	Keyboard pointer device file

NOTES

The option syntax is inconsistent with itself and *xset(1)*.

The acceleration option should take a numerator and a denominator like the protocol.

If *X* dies before its clients, new clients won't be able to connect until all existing connections have their TCP TIME_WAIT timers expire.

The color database is missing a large number of colors. However, there doesn't seem to be a better one available that can generate RGB values.

COPYRIGHT

ORIGIN

MIT Distribution

SEE ALSO

bdftosnf bitmap(1), getty(1M), gettydefs(4), gwindstop(1), hpterm(1), init(1M), inittab(4), rgb(1), uwm(1), x11start(1), xclock(1), xfc(1), xfd(1), xhost(1), xinit(1), xinitcolormap(1), xload(1), xmodmap(1), xrefresh(1), xseethru(1), xset(1), xsetroot(1), xterm(1), xwcreate(1), xwd(1), xwdestroy(1), xwininfo(1), xwud(1), *Programming With Xlib, Programming With the Xt Intrinsics*

Series 300 and 800 Only

NAME

xset - user preference utility for X

SYNOPSIS

xset [options]

DESCRIPTION

This program is used to set various user preference options of the display.

OPTIONS

-display *display*
This option specifies the server to use; see *X(1)*.

b the **b** option controls bell volume, pitch and duration. This option accepts up to three numerical parameters, a preceding dash(-), or a 'on/off' flag. If no parameters are given, or the 'on' flag is used, the system defaults will be used. If the dash or 'off' are given, the bell will be turned off. If only one numerical parameter is given, the bell volume will be set to that value, as a percentage of its maximum. Likewise, the second numerical parameter specifies the bell pitch, in hertz, and the third numerical parameter specifies the duration in milliseconds. Note that not all hardware can vary the bell characteristics. The X server will set the characteristics of the bell as closely as it can to the user's specifications.

c The **c** option controls key click. This option can take an optional value, a preceding dash(-), or an 'on/off' flag. If no parameter or the 'on' flag is given, the system defaults will be used. If the dash or 'off' flag is used, keyclick will be disabled. If a value from 0 to 100 is given, it is used to indicate volume, as a percentage of the maximum. The X server will set the volume to the nearest value that the hardware can support.

fp= *path,...*
The **fp=** sets the font path to the directories given in the path argument. The directories are interpreted by the server, not by the client, and are server-dependent. Directories that do not contain font databases created by *mkfontdir* will be ignored by the server.

fp *default*
The *default* argument causes the font path to be reset to the server's default.

fp *rehash*
The *rehash* argument causes the server to reread the font databases in the current font path. This is generally only used when adding new fonts to a font directory (after running *mkfontdir* to recreate the font database).

-fp or **fp-** *path*[*,path...*]
The **-fp** and **fp-** options remove elements from the current font path. They must be followed by a comma-separated list of directories.

+fp or **fp+** *path*[*,path...*]
This **+fp** and **fp+** options prepend and append elements to the current font path, respectively. They must be followed by a comma-separated list of directories.

led The **led** option controls the keyboard LEDs. This controls the turning on or off of one or all of the LEDs. It accepts an optional integer, a preceding dash(-) or an 'on/off' flag. If no parameter or the 'on' flag is given, all LEDs are turned on. If a preceding dash or the flag 'off' is given, all LEDs are turned off. If a value between 1 and 32 is given, that LED will be turned on or off depending on the existance of a preceding dash. A common LED which can be controlled is the "Caps Lock" LED. "xset led 3" would turn led #3 on. "xset -led 3" would turn it off. The particular LED values may refer to different LEDs on different hardware.

m [*acceleration* [*threshold*]]
The **m** option controls the mouse parameters. The parameters for the mouse are 'acceleration' and 'threshold'. The mouse, or whatever pointer the machine is connected to, will go 'acceleration' times as fast when it travels more than 'threshold' pixels in a short time. This way, the mouse can be used for precise alignment when it is moved slowly, yet it can be set to travel across the screen in a flick of the wrist when desired.

One or both parameters for the **m** option can be omitted, but if only one is given, it will be interpreted as the acceleration. If no parameters or the flag 'default' is used, the system defaults will be set.

p *pixel color*
The **p** option controls pixel color values. The parameters are the color map entry number in decimal, and a color specification. The root background colors may be changed on some servers by altering the entries for BlackPixel and WhitePixel. Although these are often 0 and 1, they need not be. Also, a server may choose to allocate those colors privately, in which case an error will be generated. The map entry must be allocated read/write, or an error will result.

r The **r** option controls the autorepeat. If a preceding dash or the 'off' flag is used, autorepeat will be disabled. If no parameters or the 'on' flag is used, autorepeat will be enabled.

s The **s** option lets you set the screen saver parameters. This option accepts up to two numerical parameters, a 'blank/noblank' flag, an 'expose/noexpose' flag, an 'on/off' flag, or the 'default' flag. If no parameters or the 'default' flag is used, the system will be set to its default screen saver characteristics. The 'on/off' flags simply turn the screen saver functions on or off. The 'blank' flag sets the preference to blank the video (if the hardware can do so) rather than display a background pattern, while 'noblank' sets the preference to display a pattern rather than blank the video. The 'expose' flag sets the preference to allow window exposures (the server can freely discard window contents), while 'noexpose' sets the preference to disable screen saver unless the server can regenerate the screens without causing exposure events. The length and period parameters for the screen saver function determines how long the server must be inactive for screen saving to activate, and the period to change the background pattern to avoid burn in. The arguments are specified in seconds. If only one numerical parameter is given, it will be used for the length.

q The **q** option gives you information on the current settings.

These settings will be reset to default values when you log out.

ENVIRONMENT

DISPLAY - To get default host and display number.

HARDWARE DEPENDENCIES

Note that not all X implementations are guaranteed to honor all of these options.

series 800

There is no hardware support for changing bell pitch or duration.

SEE ALSO

X(1), Xserver(1), xmodmap(1), xrdb(1), xsetroot(1)

COPYRIGHT

ORIGIN

MIT Distribution

Series 300 and 800 Only

NAME

xsetroot - root window parameter setting utility for X

SYNOPSIS

xsetroot [options]

DESCRIPTION

The *setroot* program allows you to tailor the appearance of the background ("root") window on a workstation display running X. Normally, you experiment with *xsetroot* until you find a personalized look that you like, then put the *xsetroot* command that produces it into your X startup file. If no options are specified, or if *-def* is specified, the window is reset to its default state. the *-def* option can be specified along with other options and only the non-specified characteristics will be reset to the default state.

Only one of the background color/tiling changing options (-solid, -gray, -grey, -bitmap, and -mod) may be specified at a time.

OPTIONS

The various options are as follows:

-help Print a usage message and exit.

-def Reset unspecified attributes to the default values. (Restores the background to the familiar gray mesh and the cursor to the hollow x shape.)

-cursor *cursorfile maskfile*
This lets you change the pointer cursor to whatever you want when the pointer cursor is outside of any window. Cursor and mask files are bitmaps (little pictures), and can be created with the *bitmap(1)* program. You probably want the mask file to be all black until you get used to the way masks work.

-bitmap *filename*
Use the bitmap specified in the file to set the window pattern. You can create your own bitmap files using the *bitmap(1)* program. The entire background will be made up of repeated "tiles" of the bitmap.

-mod *x y*
This is used to create a plaid-like grid pattern on your screen. x and y are integers ranging from 1 to 16. Zero and negative numbers are taken as 1.

-gray Make the entire background gray. (Easier on the eyes.)

-grey Make the entire background grey.

-fg *color*
Use "color" as the foreground color when setting attributes. Options that use/are affected by this parameter are -bitmap, -cursor, -mod, -gray and -grey.

-bg *color*
Use "color" as the background color when setting attributes. Options that use/are affected by this parameter are -bitmap, -cursor, -mod, -gray and -grey.

-rv This exchanges the foreground and background colors. Normally the foreground color is black and the background color is white.

-solid *color*
Set the window color to "color".

-name *string*
Set the name of the root window to "string". There is no default value. Usually a name is assigned to a window so that the window manager can use a text representation when the window is iconified. This option is unused since you can't iconify the background.

-display *display*
Specifies the server to connect to; see *X(1)*.

COPYRIGHT

ORIGIN

MIT Distribution

SEE ALSO

X(1), xset(1)

NAME

xterm - terminal emulator for X

SYNOPSIS

xterm [*-toolkitoption* ...] [-option ...]

DESCRIPTION

The *xterm* program is a terminal emulator for the X Window System. It provides DEC VT102 and Tektronix 4014 compatible terminals for programs that can't use the window system directly. If the underlying operating system supports terminal resizing capabilities (for example, the SIGWINCH signal in systems derived from 4.3bsd), *xterm* will use the facilities to notify programs running in the window whenever it is resized.

The VT102 and Tektronix 4014 terminals each have their own window so that you can edit text in one and look at graphics in the other at the same time. To maintain the correct aspect ratio (height/width), Tektronix graphics will be restricted to the largest box with a 4014's aspect ratio that will fit in the window. This box is located in the upper left area of the window.

Although both windows may be displayed at the same time, one of them is considered the "active" window for receiving keyboard input and terminal output. This is the window that contains the text cursor and whose border highlights whenever the pointer is in either window. The active window can be choosen through escape sequences, the "Modes" menu in the VT102 window, and the "Tektronix" menu in the 4014 window.

OPTIONS

The *xterm* terminal emulator accepts all of the standard X Toolkit command line options along with the additional options listed below (if the option begins with a '+' instead of a '-', the option is restored to its default value):

-132 Normally, the VT102 DECCOLM escape sequence that switches between 80 and 132 column mode is ignored. This option causes the DECCOLM escape sequence to be recognized, and the *xterm* window will resize appropriately.

-ah This option indicates that *xterm* should always highlight the text cursor and borders. By default, *xterm* will display a hollow text cursor whenever the focus is lost or the pointer leaves the window.

+ah This option indicates that *xterm* should do text cursor highlighting.

-b *number*
This option specifies the size of the inner border (the distance between the outer edge of the characters and the window border) in pixels. The default is 2.

-cc *characterclassrange:value*[,...]
This sets classes indicated by the given ranges for using in selecting by words. See the section specifying character classes.

-cr *color* This option specifies the color to use for text cursor. The default is to use the same foreground color that is used for text.

-cu This option indicates that *xterm* should work around a bug in the *curses*(3x) cursor motion package that causes the *more*(1) program to display lines that are exactly the width of the window and are followed by line beginning with a tab to be displayed incorrectly (the leading tabs are not displayed).

+cu This option indicates that that *xterm* should not work around the *curses*(3x) bug mentioned above.

-e *program* [*arguments* ...]
This option specifies the program (and its command line arguments) to be run in the *xterm* window. It also sets the window title and icon name to be the basename of the program being executed if neither *-T* nor *-n* are given on the command line. **This must be the last option on the command line.**

-fb *font* This option specifies a font to be used when displaying bold text. This font must be the same height and width as the normal font. If only one of the normal or bold fonts is specified, it will be used as the normal font and the bold font will be produced by

overstriking this font. The default bold font is "vtbold." The directory containing the font is determined by xset -fp.

-j This option indicates that *xterm* should do jump scrolling. Normally, text is scrolled one line at a time; this option allows *xterm* to move multiple lines at a time so that it doesn't fall as far behind. Its use is strongly recommended since it make *xterm* much faster when scanning through large amounts of text. The VT100 escape sequences for enabling and disabling smooth scroll as well as the "Modes" menu can be used to turn this feature on or off.

+j This option indicates that *xterm* should not do jump scrolling.

-l This option indicates that *xterm* should send all terminal output to a log file as well as to the screen. This option can be enabled or disabled using the "xterm X11" menu.

+l This option indicates that *xterm* should not do logging.

-lf *filename*
This option specifies the name of the file to which the output log described above is written. If *file* begins with a pipe symbol (|), the rest of the string is assumed to be a command to be used as the endpoint of a pipe. The default filename is "**XtermLog.***XXXXX*" (where *XXXXX* is the process id of *xterm*) and is created in the directory from which *xterm* was started (or the user's home directory in the case of a login window.

-ls This option indicates that shell that is started in the *xterm* window be a login shell (i.e. the first character of argv[0] will be a dash, indicating to the shell that it should read the user's .login or .profile).

+ls This option indicates that the shell that is started should not be a login shell (i.e. it will be normal "subshell").

-mb This option indicates that *xterm* should ring a margin bell when the user types near the right end of a line. This option can be turned on and off from the "Modes" menu.

+mb This option indicates that margin bell should not be rung.

-ms *color*
This option specifies the color to be used for the pointer cursor. The default is to use the foreground color.

-nb *number*
This option specifies the number of characters from the right end of a line at which the margin bell, if enabled, will ring. The default is 10.

-rw This option indicates that reverse-wraparound should be allowed. This allows the cursor to back up from the leftmost column of one line to the rightmost column of the previous line. This is very useful for editing long shell command lines and is encouraged. This option can be turned on and off from the "Modes" menu.

+rw This option indicates that reverse-wraparound should not be allowed.

-s This option indicates that *xterm* may scroll asynchronously, meaning that the screen does not have to be kept completely up to date while scrolling. This allows *xterm* to run faster when network latencies are very high and is typically useful when running across a very large internet or many gateways.

+s This option indicates that *xterm* should scroll synchronously.

-sb This option indicates that some number of lines that are scrolled off the top of the window should be saved and that a scrollbar should be displayed so that those lines can be viewed. This option may be turned on and off from the "Modes" menu.

+sb This option indicates that a scrollbar should not be displayed.

-sf This option indicates that Sun Function Key escape codes should be generated for function keys.

+sf This option indicates that the standard escape codes should be generated for function keys.

-si This option indicates that output to a window should not automatically reposition the screen to the bottom of the scrolling region. This option can be turned on and off from the "Modes" menu.

+si This option indicates that output to a window should cause it to scroll to the bottom.

-sk This option indicates that pressing a key while using the scrollbar to review previous lines of text should cause the window to be repositioned automatically in the normal postion at the bottom of the scroll region.

+sk This option indicates that pressing a key while using the scrollbar should not cause the window to be repositioned.

-sl *number*
This option specifies the number of lines to save that have been scrolled off the top of the screen. The default is 64.

-t This option indicates that *xterm* should start in Tektronix mode, rather than in VT102 mode. Switching between the two windows is done using the "Modes" menus.

+t This option indicates that *xterm* should start in VT102 mode.

-vb This option indicates that a visual bell is prefered over an audible one. Instead of ringing the terminal bell whenever a Control-G is received, the window will be flashed.

+vb This option indicates that a visual bell should not be used.

-C This option indicates that this window should be receive console output. This is not supported on all systems.

-L This option indicates that *xterm* was started by *init*. In this mode, *xterm* does not try to allocate a new pseudoterminal as *init* has already done so. In addition, the system program *getty* is run instead of the user's shell. **This option should never be used by users when starting terminal windows.**

-S*ccn* This option specifies the last two letters of the name of a pseudoterminal to use in slave mode. This allows *xterm* to be used as an input and output channel for an existing program and is sometimes used in specialized applications.

The following command line arguments are provided for compatibility with older versions. They may not be supported in the next release as the X Toolkit provides standard options that accomplish the same task.

%geom This option specifies the prefered size and position of the Tektronix window. It is shorthand for specifying the "**tekGeometry*" resource.

#geom This option specifies the prefered position of the icon window. It is shorthand for specifying the "**iconGeometry*" resource.

-T *string* This option specifies the title for *xterm*'s windows. It is equivalent to **-title**.

-n*string* This option specifies the icon name for *xterm*'s windows. It is shorthand for specifying the "**iconName*" resource.

-r This option indicates that reverse video should be simulated by swapping the foreground and background colors. It is equivalent to **-reversevideo** or **-rv**.

-w *number*
This option specifies the width in pixels of the border surrounding the window. It is equivalent to **-borderwidth** or **-bw**.

The following standard X Toolkit command line arguments are commonly used with *xterm*:

-bg *color*
This option specifies the color to use for the background of the window. The default is "white."

-bd *color*
This option specifies the color to use for the border of the window. The default is "black."

-bw *number*
This option specifies the width in pixels of the border surrounding the window.

-fg *color* This option specifies the color to use for displaying text. The default is "black".

-fn *font* This option specifies the font to be used for displaying normal text. The default is "vtsingle." The directory containing the font is determined by xset -fp.

-name *name*
This option specifies the application name under which resource are to be obtained, rather than the default executable file name.

-rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

-geometry *geometry*
This option specifies the prefered size and position of the VT102 window; see *X(1)*;

-display *display*
This option specifies the X server to contact; see *X(1)*.

-xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

-iconic This option indicates that *xterm* should ask the window manager to start it as an icon rather than as the normal window.

X DEFAULTS

The program understands all of the core X Toolkit resource names and classes as well as:

name (class **Name**)
Specifies the name of this instance of the program. The default is "xterm."

iconGeometry (class **IconGeometry**)
Specifies the prefered size and position of the application when iconified. It is not necessarily obeyed by all window managers.

title (class **Title**)
Specifies a string that may be used by the window manager when displaying this application.

utmpInhibit (class **UtmpInhibit**)
Specifies whether or not *xterm* should try to record the user's terminal in */etc/utmp*.

sunFunctionKeys (class **SunFunctionKeys**)
Specifies whether or not Sun Function Key escape codes should be generated for function keys instead of standard escape sequences.

The following resources are specified as part of the "vt100" widget (class "VT100"):

alwaysHighlight (class **AlwaysHighlight**)
Specifies whether or not *xterm* should always display a highlighted text cursor. By default, a hollow text cursor is displayed whenever the pointer moves out of the window or the window loses the input focus.

font (class **Font**)
Specifies the name of the normal font. The default is "vtsingle."

boldFont (class **Font**)
Specifies the name of the bold font. The default is "vtbold."

c132 (class **C132**)
Specifies whether or not the VT102 DECCOLM escape sequence should be honored. The default is "false."

charClass (class **CharClass**)
Specifies comma-separated lists of character class bindings of the form *[low-]high:value*. These are used in determining which sets of characters should be treated the same when doing cut and paste. See the section on specifying character classes.

curses (class **Curses**)
Specifies whether or not the last column bug in cursor should be worked around. The default is "false."

background (class **Background**)
Specifies the color to use for the background of the window. The default is "white."

foreground (class **Foreground**)
Specifies the color to use for displaying text in the window. Setting the class name instead of the instance name is an easy way to have everything that would normally appear in the "text" color change color. The default is "black."

cursorColor (class **Foreground**)
Specifies the color to use for the text cursor. The default is "black."

geometry (class **Geometry**)
Specifies the prefered size and position of the VT102 window.

tekGeometry (class **Geometry**)
Specifies the prefered size and position of the Tektronix window.

internalBorder (class **BorderWidth**)
Specifies the number of pixels between the characters and the window border. The default is 2.

jumpScroll (class **JumpScroll**)
Specifies whether or not jump scroll should be used. The default is "false".

logFile (class **Logfile**)
Specifies the name of the file to which a terminal session is logged. The default is "**XtermLog.***XXXXX*" (where *XXXXX* is the process id of *xterm*).

logging (class **Logging**)
Specifies whether or not a terminal session should be logged. The default is "false."

logInhibit (class **LogInhibit**)
Specifies whether or not terminal session logging should be inhibited. The default is "false."

loginShell (class **LoginShell**)
Specifies whether or not the shell to be run in the window should be started as a login shell. The default is "false."

marginBell (class **MarginBell**)
Specifies whether or not the bell should be run when the user types near the right margin. The default is "false."

multiScroll (class **MultiScroll**)
Specifies whether or not asynchronous scrolling is allowed. The default is "false."

nMarginBell (class **Column**)
Specifies the number of characters from the right margin at which the margin bell should be run, when enabled.

pointerColor (class **Foreground**)
Specifies the color of the pointer. The default is "black."

pointerShape (class **Cursor**)
Specifies the name of the shape of the pointer. The default is "xterm."

reverseVideo (class **ReverseVideo**)
Specifies whether or not reverse video should be simulated. The default is "false."

reverseWrap (class **ReverseWrap**)
Specifies whether or not reverse-wraparound should be enabled. The default is "false."

saveLines (class **SaveLines**)
Specifies the number of lines to save beyond the top of the screen when a scrollbar is turned on. The default is 64.

scrollBar (class **ScrollBar**)
Specifies whether or not the scrollbar should be displayed. The default is "false."

scrollInput (class **ScrollCond**)
Specifies whether or not output to the terminal should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "true."

scrollKey (class **ScrollCond**)
Specifies whether or not pressing a key should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "false."

signalInhibit (class **SignalInhibit**)
Specifies whether or not the entries in the "xterm X11" menu for sending signals to *xterm* should be disallowed. The default is "false."

tekInhibit (class **TekInhibit**)
Specifies whether or not Tektronix mode should be disallowed. The default is "false."

tekStartup (class **TekStartup**)
Specifies whether or not *xterm* should start up in Tektronix mode. The default is "false."

titeInhibit (class **TiteInhibit**)
Specifies whether or not *xterm* should remove remove *ti* or *te* termcap entries (used to switch between alternate screens on startup of many screen-oriented programs) from the TERMCAP string.

visualBell (class **VisualBell**)
Specifies whether or not a visible bell (i.e. flashing) should be used instead of an audible bell when Control-G is received. The default is "false."

The following resources are specified as part of the "tek4014" widget (class "Tek4014"):

width (class **Width**)
Specifies the width of the Tektronix window in pixels.

height (class **Height**)
Specifies the height of the Tektronix window in pixels.

The following resources are specified as part of the "menu" widget:

menuBorder (class **MenuBorder**)
Specifies the size in pixels of the border surrounding menus. The default is 2.

menuFont (class **Font**)
Specifies the name of the font to use for displaying menu items.

menuPad (class **MenuPad**)
Specifies the number of pixels between menu items and the menu border. The default is 3.

The following resources are useful when specified for the Athena Scrollbar widget:

thickness (class **Thickness**)
Specifies the width in pixels of the scrollbar.

background (class **Background**)
Specifies the color to use for the background of the scrollbar.

EMULATIONS
The VT102 emulation is fairly complete, but does not support the blinking character attribute nor

the double-wide and double-size character sets. *Termcap*(5) entries that work with *xterm* include "xterm", "vt102", "vt100" and "ansi", and *xterm* automatically searches the termcap file in this order for these entries and then sets the "TERM" and the "TERMCAP" environment variables.

Many of the special *xterm* features (like logging) may be modified under program control through a set of escape sequences different from the standard VT102 escape sequences.

The Tektronix 4014 emulation is also fairly good. Four different font sizes and five different lines types are supported. The Tektronix text and graphics commands are recorded internally by *xterm* and may be written to a file by sending the COPY escape sequence (or through the **Tektronix** menu; see below). The name of the file will be "COPY*yy-MM-dd.hh:mm:ss*", where *yy*, *MM*, *dd*, *hh*, *mm* and *ss* are the year, month, day, hour, minute and second when the COPY was performed (the file is created in the directory *xterm* is started in, or the home directory for a login *xterm*).

POINTER USAGE

Once the VT102 window is created, *xterm* allows you to select text and copy it within the same or other windows.

The selection functions are invoked when the pointer buttons are used with no modifiers, and when they are used with the "shift" key.

Pointer button one (usually left) is used to save text into the cut buffer. Move the cursor to beginning of the text, and then hold the button down while moving the cursor to the end of the region and releasing the button. The selected text is highlighted and is saved in the global cut buffer when the button is released. Double-clicking selects by words. Triple-clicking selects by lines. Quadruple-clicking goes back to characters, etc. Multiple-click is determined by the time from button up to button down, so you can change the selection unit in the middle of a selection.

Pointer button two (usually middle) 'types' (pastes) the text from the cut buffer, inserting it as keyboard input.

Pointer button three (usually right) extends the current selection. (Without loss of generality, that is you can swap "right" and "left" everywhere in the rest of this paragraph...) If pressed while closer to the right edge of the selection than the left, it extends/contracts the right edge of the selection. If you contract the selection past the left edge of the selection, *xterm* assumes you really meant the left edge, restores the original selection, then extends/contracts the left edge of the selection. Extension starts in the selection unit mode that the last selection or extension was performed in; you can multiple-click to cycle through them.

By cutting and pasting pieces of text without trailing new lines, you can take text from several places in different windows and form a command to the shell, for example, or take output from a program and insert it into your favorite editor. Since the cut buffer is globally shared among different applications, you should regard it as a 'file' whose contents you know. The terminal emulator and other text programs should be treating it as if it were a text file, i.e. the text is delimited by new lines.

The scroll region displays the position and amount of text currently showing in the window (highlighted) relative to the amount of text actually saved. As more text is saved (up to the maximum), the size of the highlighted area decreases.

Clicking button one with the pointer in the scroll region moves the adjacent line to the top of the display window.

Clicking button three moves the top line of the display window down to the pointer position.

Clicking button two moves the display to a position in the saved text that corresponds to the pointer's position in the scrollbar.

Unlike the VT102 window, the Tektronix window does not allow the copying of text. It does allow Tektronix GIN mode, and in this mode the cursor will change from an arrow to a cross. Pressing any key will send that key and the current coordinate of the cross cursor. Pressing button one, two, or three will return the letters 'l', 'm', and 'r', respectively. If the 'shift' key is pressed when a pointer buton is pressed, the corresponding upper case letter is sent. To distinguish a pointer button from a key, the high bit of the character is set (but this is bit is normally stripped unless the terminal mode is RAW; see *tty*(4) for details).

MENUS

Xterm has three different menus, named **xterm**, **Modes**, and **Tektronix**. Each menu pops up under the correct combinations of key and button presses. Most menus are divided into two section, separated by a horizontal line. The top portion contains various modes that can be altered. A check mark appears next to a mode that is currently active. Selecting one of these modes toggles its state. The bottom portion of the menu are command entries; selecting one of these performs the indicated function.

The **xterm** menu pops up when the "control" key and pointer button one are pressed in a window. The modes section contains items that apply to both the VT102 and Tektronix windows. Notable entries in the command section of the menu are the **Continue**, **Suspend**, **Interrupt**, **Hangup**, **Terminate** and **Kill** which sends the SIGCONT, SIGTSTP, SIGINT, SIGHUP, SIGTERM and SIGKILL signals, respectively, to the process group of the process running under *xterm* (usually the shell). The **Continue** function is especially useful if the user has accidentally typed CTRL-Z, suspending the process.

The **Modes** menu sets various modes in the VT102 emulation, and is popped up when the "control" key and pointer button two are pressed in the VT102 window. In the command section of this menu, the soft reset entry will reset scroll regions. This can be convenient when some program has left the scroll regions set incorrectly (often a problem when using VMS or TOPS-20). The full reset entry will clear the screen, reset tabs to every eight columns, and reset the terminal modes (such as wrap and smooth scroll) to there initial states just after *xterm* has finish processing the command line options. The **Tektronix** menu sets various modes in the Tektronix emulation, and is popped up when the "control" key and pointer button two are pressed in the Tektronix window. The current font size is checked in the modes section of the menu. The **PAGE** entry in the command section clears the Tektronix window.

CHARACTER CLASSES

Clicking the middle mouse button twice in rapid succession will cause all characters of the same class (e.g. letters, white space, punctuation) to be selected. Since different people have different preferences for what should be selected (for example, should filenames be selected as a whole or only the separate subnames), the default mapping can be overridden through the use of the *clarClass* (class *CharClass*) resource.

This resource is simply a list of *range:value* pairs where the range is either a single number or *low-high* in the range of 0 to 127, corresponding to the ASCII code for the character or characters to be set. The *value* is arbitrary, although the default table uses the character number of the first character occurring in the set.

The default table is:

```
static int charClass[128] = {
/* NUL SOH STX ETX EOT ENQ ACK BEL */
   32,  1,  1,  1,  1,  1,  1,  1,
/*  BS  HT  NL  VT  NP  CR  SO  SI */
    1, 32,  1,  1,  1,  1,  1,  1,
/* DLE DC1 DC2 DC3 DC4 NAK SYN ETB */
    1,  1,  1,  1,  1,  1,  1,  1,
/* CAN  EM SUB ESC  FS  GS  RS  US */
    1,  1,  1,  1,  1,  1,  1,  1,
/*  SP   !   "   #   $   %   &   ' */
   32, 33, 34, 35, 36, 37, 38, 39,
/*   (   )   *   +   ,   -   .   / */
   40, 41, 42, 43, 44, 45, 46, 47,
/*   0   1   2   3   4   5   6   7 */
   48, 48, 48, 48, 48, 48, 48, 48,
/*   8   9   :   ;   <   =   >   ? */
   48, 48, 58, 59, 60, 61, 62, 63,
/*   @   A   B   C   D   E   F   G */
   64, 48, 48, 48, 48, 48, 48, 48,
/*   H   I   J   K   L   M   N   O */
```

```
  48, 48, 48, 48, 48, 48, 48, 48,
/* P  Q  R  S  T  U  V  W */
  48, 48, 48, 48, 48, 48, 48, 48,
/* X  Y  Z  [  \  ]  ^  _ */
  48, 48, 48, 91, 92, 93, 94, 48,
/* `  a  b  c  d  e  f  g */
  96, 48, 48, 48, 48, 48, 48, 48,
/* h  i  j  k  l  m  n  o */
  48, 48, 48, 48, 48, 48, 48, 48,
/* p  q  r  s  t  u  v  w */
  48, 48, 48, 48, 48, 48, 48, 48,
/* x  y  z  {  |  }  ~ DEL */
  48, 48, 48,123,124,125,126,  1};
```

For example, the string "33:48,37:48,46-47:48,64:48" indicates that the exclamation mark, percent sign, period, slash, and ampersand characters should be treated the same way as characters and numbers. This is very useful for cutting and pasting electronic mailing addresses and filenames.

OTHER FEATURES

Xterm automatically highlights the window border and text cursor when the pointer enters the window (selected) and unhighlights them when the pointer leaves the window (unselected). If the window is the focus window, then the window is highlighted no matter where the pointer is.

In VT102 mode, there are escape sequences to activate and deactivate an alternate screen buffer, which is the same size as the display area of the window. When activated, the current screen is saved and replace with the alternate screen. Saving of lines scrolled off the top of the window is disabled until the normal screen is restored. The *termcap*(5) entry for *xterm* allows the visual editor *vi*(1) to switch to the alternate screen for editing, and restore the screen on exit.

In either VT102 or Tektronix mode, there are escape sequences to change the name of the windows and to specify a new log file name.

ENVIRONMENT

Xterm sets the environment variables "TERM" and "TERMCAP" properly for the size window you have created. It also uses and sets the environment variable "DISPLAY" to specify which bit map display terminal to use. The environment variable "WINDOWID" is set to the X window id number of the *xterm* window.

NOTES

Xterm will hang forever if you try to paste too much text at one time. It is both producer and consumer for the pty and can deadlock.

Variable-width fonts are not handled reasonably.

This program still needs to be rewritten. It should be split into very modular sections, with the various emulators being completely separate widgets that don't know about each other. Ideally, you'd like to be able to pick and choose emulator widgets and stick them into a single control widget.

The focus is considered lost if some other client (e.g., the window manager) grabs the pointer; it is difficult to do better without an addition to the protocol.

There needs to be a dialog box to allow entry of log file name and the COPY file name.

Many of the options are not resettable after *xterm* starts.

This manual page is too long. There should be a separate users manual defining all of the non-standard escape sequences.

All programs should be written to use X directly; then we could eliminate this program.

COPYRIGHT

ORIGIN

MIT Distribution

SEE ALSO

resize(1), X(1), pty(4), tty(4)

NAME

xwcreate - create a new X window

SYNOPSIS

xwcreate [options] name

DESCRIPTION

This command creates a new X window and assigns it the name *name.* This program will also create a *pty* file of the same name as the window in the indicated directory. After the window has been created, the *pty* file may be used to specify the window that an application should use when utilizing a graphics library (e.g. Starbase or HP-GKS). A window created by *xwcreate* can be destroyed by *xwdestroy(1).* Note: The window is actually created and maintained by the daemon program, *gwind.* The *xwcreate* program requests the daemon to create the window. If *gwind* is not running when *xwcreate* is executed, *xwcreate* will start *gwind.*

OPTIONS

-display *display*

Specifies the server to connect to; See *X(1)* for details. A limitation in xwcreate requires that the display name must be no more than 20 characters long.

-parent *parent*

Name of the window which is to be the parent of *name.* If named, the parent window must have been created by a previous invocation of *xwcreate* and must not have been destroyed by *xwdestroy(1);* otherwise an error message will be generated. If parent window is not named, the RootWindow of the display and screen will be used as the parent. If specified, parent window's name must be no more than 12 characters long.

-geometry *geometry*

This option specifies the preferred size and position of the window; See *X(1)* for details.

-r Requests the X server to create backing store for the window. By default, windows are not created with backing store.

-bg *color*

This option specifies the background color. By default, background color of the window will be black.

-bw *pixels*

This option specifies the width in pixels of the window border. By default, border of the window will be 3 pixels wide.

-bd *color*

This option specifies the border color. By default, the window border will be white.

-depth *depth*

This option specifies the visual depth of the window. By default, the window will have the same depth as its parent. If the specified depth is not supported by the display, an error will be generated and the window will not be created.

-wmdir *directory*

is the name of the directory where the pty file for the window will be created. If this option is not defined, then the directory name will be computed as follows: first, the environment of the process will be searched for the variable $WMDIR. If the variable $WMDIR is defined in the environment, then it will be used as the desired directory. If the the variable $WMDIR is not defined in the environment, then the pty file will be created in the default directory "/dev/screen". If the option *-wmdir* is defined in the command line, the directory name will be obtained as follows: If the directory argument implies an absolute pathname, then it will be taken to be the desired directory. Otherwise, the directory name will be taken to be relative to the value of the environment variable $WMDIR. If $WMDIR is not defined in the environment, the directory name will be taken to be relative to */dev/screen.* Note: if $WMDIR is defined in the environment, it must represent an absolute pathname. if *-wmdir* is defined in the command line, then the implied directory must have already been created. Otherwise, an error ("Invalid

directory") will be generated.

-title *name*
is the name to be used to reference the window. The *name* must be no more than 12 characters long.

X DEFAULTS

xwcreate uses the Xlib routine *XGetDefault(3X)* to read its Xdefaults, so its resource names are all capitalized.

Background
Specifies the window's background color.

BorderColor
Specifies the border color. This option is useful only on color displays.

BorderWidth
Specifies the border width.

Depth Specifies the visual depth of the created window.

Retained
If 'on', requests the X server to create backing store for the window.

Wmdir Specifies the default directory where the pty file will be created. See *-wmdir* above for details.

Geometry
Specifies the default positioning and/or sizing for the created window. See *X(1)* for details.

EXAMPLES

xwcreate FullView
Create a window named "FullView". Since no other argument is provided, the default geometry, border color, etc. of FullView will be taken from the RootWindow of the window's display and screen.

xwcreate HalfView -display remote_host:1.2 -parent FullView
-geometry 400x200+5+10 -r -bw 10
Create a window named "HalfView" on the display "remote_host:1.2". HalfView will be a child of the window "FullView". The upper left hand corner of HalfView will be located at coordinate 5,10 of FullView and will be 400 pixels wide and 200 pixels high. The border of HalfView will be 10 pixels wide and the border colors will be the same as FullView.

ENVIRONMENT

DISPLAY - the default host and display number.
WMDIR - the window manager directory.
/dev/screen - the default window manager directory.

DIAGNOSTICS

If the window is created successfully, *xwcreate* will remain silent. Otherwise *xwcreate* prints one or more error messages to standard output. For example:

No such display.
Named window exists.
Named parent window does not exist.
Couldn't communicate with gwind.

NOTES

If *-wmdir* is used or *WMDIR* environment variable is set to
other than the default, the directory used must exist on the same physical device as "/dev".

The WM_CLASS of an xwcreate'ed window is **Xwcreate**.

If XKillClient is used (used by some window managers) on one
of the windows created by xwcreate, all xwcreate windows (started with the same "-display" argument) will also be destroyed.

ORIGIN

HP

SEE ALSO

X(1), XOpenDisplay(3x), xwdestroy(1).

Series 300 and 800 Only

NAME

xwd - dump an image of an X window

SYNOPSIS

xwd [options]

DESCRIPTION

xwd is an X Window System window dumping utility. *xwd* allows X users to store window images in a specially formatted dump file. This file can then be read by various other X utilities for redisplay, printing, editing, formatting, archiving, image processing etc.. The target window is selected by clicking the mouse in the desired window. The keyboard bell is rung once at the beginning of the dump and twice when the dump is completed.

OPTIONS

-help Print out the 'Usage:' command syntax summary.

-id *id* This option allows the user to specify a target window *id* on the command line rather than using the mouse to select the target window.

-name *name*
This option allows the user to specify that the window named *name* is the target window on the command line rather than using the mouse to select the target window.

-root This option specifies that X's root window is the target window.

-nobdrs This argument specifies that the window dump should not include the pixels that compose the X window border. This is useful in situations where you may wish to include the window contents in a document as an illustration.

-out *filename*
This argument allows the user to explicitly specify the output file on the command line. The default is to output to standard out.

-xy This option applies to color displays only. It selects 'XY' format dumping instead of the default 'Z' format.

-add *value*
This option specifies an signed value to be added to every pixel.

-display *display*
Specifies the server to connect to; see *X(1)*.

ENVIRONMENT

DISPLAY
To get default host and display number.

NOTES

Xwd cannot get the image of child windows with a different number of planes than the specified window.

FILES

XWDFile.h
X Window Dump File format definition file.

COPYRIGHT

ORIGIN

MIT Distribution

SEE ALSO

xwud(1), X(1)

Series 300 and 800 Only

NAME

xwd2sb - translate xwd bitmap to Starbase bitmap format

SYNOPSIS

xwd2sb

DESCRIPTION

This command translates a bitmap file created by the *xwd(1)* X window dump utility program into a Starbase bitmap file as described in *bitmapfile(4)*. Translation is done from standard input to standard output.

Bitmaps created by *xwd* in the *XYPixmap* format are translated into *plane-major full-depth* Starbase bitmaps. *ZPixmap* format bitmaps are translated into *pixel-major* Starbase bitmaps.

Xwd format bitmaps with visual class *TrueColor* or *DirectColor* are translated into Starbase bitmaps with the colormap mode *CMAP_FULL*. Other visual classes result in Starbase bitmaps with the *CMAP_NORMAL* colormap mode.

Window borders stored by *xwd* are stripped from the image during translation.

OPTIONS

none

EXAMPLES

xwd | xwd2sb | pcltrans | lp -oraw

Invokes *xwd* to dump the contents of a window in *ZPixmap* format, *xwd2sb* translates the window image into Starbase format, *pcltrans* prepares the image for printing, and *lp* spools the image for the printer.

xwd -xy | xwd2sb > sbimage

Invokes *xwd* to dump the contents of a window in *XYPixmap* format and *xwd2sb* to translate the image into Starbase plane-major full-depth format. The Starbase bitmap image is placed in the *sbimage* file. (Note that *pcltrans* is unable to process plane-major full-depth images.)

xwd2sb <xwdfile >sbfile

Translates the image in *xwdfile* to Starbase format and places the result in *sbfile.*

RESTRICTIONS

XWD bitmaps must be 1-8, 12, or 24 planes deep. Bitmaps of depth 1-8 may have a visual class of GrayScale, StaticGray, PseudoColor, or StaticColor. Bitmaps of depths 12 or 24 must be of the DirectColor or TrueColor visual class.

A 12 plane bitmap must have four bits each for red, green, and blue. A 24 plane bitmap must have eight bits each for red, green, and blue.

ORIGIN

Hewlett-Packard GTD

SEE ALSO

xwd(1), pcltrans(1), bitmapfile(4).

Starbase Graphics Techniques, HP-UX Concepts and Tutorials, chapters on "Color" and "Storing and Printing Images".

Series 300 and 800 Only

NAME

xwdestroy - destroy one or more existing windows

SYNOPSIS

xwdestroy [-wmdir *directory*] *window1 window2* **...**

DESCRIPTION

If a window named in the list was created using *xwcreate(1)*, then it is destroyed, along with its children. Also the *pty* devices associated with these windows are removed. Window names may not be more than 12 characters long.

-wmdir *directory*

is the name of the directory where the pty file for the window was created. If this option is not defined, then the directory name will be computed as follows: first, the environment of the process will be searched for the variable $WMDIR. If the variable $WMDIR is defined in the environment, then it will be used as the desired directory. If the variable $WMDIR is not defined in the environment, then the pty file will be destroyed in the default directory "/dev/screen". If the option *-wmdir* is defined in the command line, the directory name will be obtained as follows: If the directory argument implies an absolute pathname, then it will be taken to be the desired directory. Otherwise, the directory name will be taken to be relative to the value of the environment variable $WMDIR. If $WMDIR is not defined in the environment, the directory name will be taken to be relative to */dev/screen*. Note: if $WMDIR is defined in the environment, it must represent an absolute pathname. If *-wmdir* is defined in the command line, then the implied directory must have already been created. Otherwise, an error ("Invalid directory") will be generated.

ENVIRONMENT

WMDIR - the window manager directory.
/dev/screen - the default window manager directory.

DIAGNOSTICS

If the windows were destroyed successfully, the program remains silent. If one or more of the windows could not be destroyed because of some error, appropriate message will be printed on standard output. For example:

Invalid directoy
Named window does not exist.

ORIGIN

Hewlett-Packard Company

SEE ALSO

XOpenDisplay(3), xwcreate(1).

Series 300 and 800 Only

NAME

xwininfo - window information utility for X

SYNOPSIS

xwininfo [options]

DESCRIPTION

xwininfo is a utility for displaying information about windows. Various information is displayed depending on which options are selected. If no options are chosen, **-stats** is assumed.

The user has the option of selecting the target window with the mouse (by clicking any mouse button in the desired window) or by specifying its window id on the command line with the **-id** option. Or instead of specifying the window by its id number, the **-name** option may be used to specify which window is desired by name. There is also a special **-root** option to quickly obtain information on X's root window.

OPTIONS

-help Print out the 'Usage:' command syntax summary.

-id *id* This option allows the user to specify a target window *id* on the command line rather than using the mouse to select the target window. This is very useful in debugging X applications where the target window is not mapped to the screen or where the use of the mouse might be impossible or interfere with the application.

-name *name*
This option allows the user to specify that the window named *name* is the target window on the command line rather than using the mouse to select the target window.

-root This option specifies that X's root window is the target window. This is useful in situations where the root window is completely obscured.

-int This option specifies that all X window ids should be displayed as integer values. The default is to display them as hexadecimal values.

-tree This option causes the root, parent, and immediate child windows' ids and names of the selected window to be displayed.

-recurs This option causes the root, parent, and all progeny windows' ids and names of the selected window to be displayed. It is a recursive version of the -tree option.

-stats This option causes the display of various attributes pertaining to the location and appearance of the selected window. Information displayed includes the location of the window, its width and height, its depth, border width, class, colormap id if any, map state, backing-store hint, and location of the corners.

-bits This option causes the display of various attributes pertaining to the selected window's raw bits and how the selected window is to be stored. Displayed information includes the selected window's bit gravity, window gravity, backing-store hint, backing-planes value, backing pixel, and whether or not the window has save-under set.

-events This option causes the selected window's event masks to be displayed. Both the event mask of events wanted by some client and the event mask of events not to propagate are displayed.

-size This option causes the selected window's sizing hints to be displayed. Displayed information includes: for both the normal size hints and the zoom size hints, the user supplied location if any; the program supplied location if any; the user supplied size if any; the program supplied size if any; the minimum size if any; the maximum size if any; the resize increments if any; and the minimum and maximum aspect ratios if any.

-wm This option causes the selected window's window manager hints to be displayed. Information displayed may include whether or not the application accepts input, what the window's icon window # and name is, where the window's icon should go, and what the window's initial state should be.

-metric This option causes all individual height, width, and x and y positions to be displayed in millimeters as well as number of pixels, based on what the server thinks the resolution is.

Geometry specifications that are in +x+y form are not changed.

-english This option causes all individual height, width, and x and y positions to be displayed in inches (and feet, yards, and miles if necessary) as well as number of pixels. **-metric** and **-english** may both be enabled at the same time.

-all This option is a quick way to ask for all information possible.

-display *display*
This option allows you to specify the server to connect to; see *X(1)*.

EXAMPLE

The following is a sample summary taken with no options specified:

```
xwininfo = = > Please select the window about which you
     = = > would like information by clicking the
     = = > mouse in that window.

xwininfo = = > Window id: 0x60000f (xterm)

     = = > Upper left X: 4
     = = > Upper left Y: 19
     = = > Width: 726
     = = > Height: 966
     = = > Depth: 4
     = = > Border width: 3
     = = > Window class: InputOutput
     = = > Colormap: 0x80065
     = = > Window Bit Gravity State: NorthWestGravity
     = = > Window Window Gravity State: NorthWestGravity
     = = > Window Backing Store State: NotUseful
     = = > Window Save Under State: no
     = = > Window Map State: IsViewable
     = = > Window Override Redirect State: no
     = = > Corners:  +4+19  -640+19  -640-33  +4-33
```

ENVIRONMENT

DISPLAY
To get the default host and display number.

SEE ALSO

X(1), xprop(1)

NOTES

Using **-stats -bits** shows some redundant information.

COPYRIGHT

ORIGIN

MIT Distribution

Series 300 and 800 Only

NAME

xwud - image displayer for X

SYNOPSIS

xwud [-in *file*] [-geometry *geom*] [-display *display*] [-new] [-raw] [-help] [-rv] [-plane *number*] [-fg *color*] [-bg *color*]

DESCRIPTION

Xwud is an X Window System image undumping utility. *Xwud* allows X users to display in a window an image saved in a specially formatted dump file, such as produced by *xwd(1)*.

Clicking any button in the window will terminate the application.

OPTIONS

-help Print out a short description of the allowable options.

-in *file* This option allows the user to explicitly specify the input file on the command line. If no input file is given, the standard input is assumed.

-rv If a bitmap image (or a single plane of an image) is displayed, this option forces the foreground and background colors to be swapped. This may be needed when displaying a bitmap image which has the color sense of pixel values "0" and "1" reversed from what they are on your display.

-display *display*
This option allows you to specify the server to connect to; see *X(1)*.

-geometry *geom*
This option allows you to specify the size and position of the window. Typically you will only want to specify the position, and let the size default to the actual size of the image.

-new This option forces creation of a new colormap for displaying the image. If the image characteristics happen to match those of the display, this can get the image on the screen faster, but at the cost of using a new colormap (which on most displays will cause other windows to go technicolor).

-raw This option forces the image to be displayed with whatever color values happen to currently exist on the screen. This option is mostly useful when undumping an image back onto the same screen that the image originally came from, while the original windows are still on the screen, and results in getting the image on the screen faster.

-plane *number*
You can select a single bit plane of the image to display with this option. Planes are numbered with zero being the least significant bit. This option can be used to generate a single plane image to pass to *xpr(1)* for printing.

-fg *color* If a bitmap image (or a single plane of an image) is displayed, this option can be used to specify the color to display for the "1" bits in the image.

-bg *color*
If a bitmap image (or a single plane of an image) is displayed, this option can be used to specify the color to display for the "0" bits in the image.

ENVIRONMENT

DISPLAY
To get default display.

FILES

XWDFile.h
X Window Dump File format definition file.

NOTES

Needs to be faster when translating colors. Needs to be much faster when translating more than 256 DirectColor colors. Although *xwud* will display across visual types and depths, if the image contains more colors than the colormap of the displaying visual, results will be undefined.

SEE ALSO

xwd(1), xpr(1), X(1)

COPYRIGHT

ORIGIN

MIT Distribution

NAME

bdf - Bitmap Distribution Format 2.1

DESCRIPTION

A "bdf" file is a file conforming to Bitmap Distribution Format 2.1. It is used for specifying fonts for the X11 Windowing System. In bdf format, the font is transportable between systems and is converted to a server natural format via the use of **bdftosnf** (see bdftosnf(1) for more details).

FILE FORMAT

STARTFONT 2.1
This must be the first line of the file.

COMMENT
This must be the second line in the file and can be 1 or more lines. These lines are ignored by the font compiler.

FONT *family_name-face_name*
This must be the third line in the file. Examples of a family name are "oldenglish", "6x10.bits", "germanic". Examples of face name are "bold" and "italic".

SIZE *point_size x_resolution y_resolution*
This must be the fourth line in the file. *point_size* is usually based on 72 points per inch. If a font has a point size of 8, it will be 1/9th of an inch high; point size of 16 will be 1/4 of an inch high. The *x_resolution* and *y_resolution* is the pixels per inch on the display for which the font was created.

FONTBOUNDINGBOX *width height x_displacement y_displacement*
This must be the fifth line in the file. *width* and *height* are the maximum width and height in pixels of the font. *x_displacement* and *y_displacement* indicate (again in pixels) the lower left corner of the font with respect to the origin.

STARTPROPERTIES *p*
This is an optional keyword. If included, *p* indicates the number of special properties following it. See "SPECIAL PROPERTIES" for a complete description of the special properties available.

ENDPROPERTIES
If there is a **STARTPROPERTIES**, there must be an **ENDPROPERTIES**.

CHARS *c*
This keyword must be the next statement after **FONTBOUNDINGBOX** (or **ENDPROPERTIES** if special properties are declared). *c* indicates the number of characters the font will contain. **CHARS** is followed by *c* **STARTCHAR, ENDCHAR** pairs. See "CHARACTER DEFINITION" for details on the information supplied by the **STARTCHAR** and **ENDCHAR** pair.

ENDFONT
This must be the last line in the file. It indicates the end of the information the font compiler is to process.

CHARACTER DEFINITION

STARTCHAR *name*
This is the first keyword for each character to be described for the font. *name* is the descriptive name of the glyph, i.e. "j", "x", "1", etc.

ENCODING *integer*
This must be the first keyword to follow **STARTCHAR**. It indicates the Adobe Standard Encoding value for *name*. *integer* must be a positive integer. If it is a -1, it is assumed that the character is not a member of the Adobe Standard Encoding. In this case, the -1 can be followed by an optional integer specifying the glyph index.

SWIDTH *scalable_width_x scalable_width_y*
This is the scalable width of the character in x and y. To calculate the the width in device pixels from the scalable width, multiply **SWIDTH** by *p* times *r* divided by 72000 where *p* is the size of the character in points and *r* is the device resolution in pixels per inch.

DWIDTH *width_x width_y*
width_x and *width_y* are a vector in device units indicating the position of the next character's origin relative to this character.

BBX *BBw BBh BBox BBoy*
BBw and *BBh* are the width and height of this character. *BBox* and *BBoy* are the x and y displacement from the origin to the lower left corner of the character.

BITMAP
This keyword appears on the line by itself and is followed by *BBh* lines of hex-coded bitmap, padded on the right with zeros to the nearest multiple of 8.

ENDCHAR
For each **STARTCHAR**, there must be an **ENDCHAR**.

SPECIAL PROPERTIES
The special properties information is saved in the special properties list of the font for access by any application wishing more information about the font. The currently supported special properties are:

FONT_ASCENT *integer*
Determines the top boundary of the font.

FONT_DESCENT *integer*
Determines the bottom boundary of the font.

DEFAULT_CHAR *integer*
This is the default character to use for all undefined characters.

POINT_SIZE *integer*
Same as *point_size* specified in **SIZE**.

FAMILY_NAME *string*
Same as *family_name* specified in **FONT**.

RESOLUTION *integer*
The resolution in pixels per inch of the display for which the font was created.

X_HEIGHT *integer*
This specifies the height of the lower case "x" in quarter-dot units (four quarter-dot units equals one pixel).

WEIGHT *integer*
Specifies the thickness of the strokes used in designing the font. The range is -7 to 7 (thin - thick).

ORIGIN
Character Bitmap Distribution Format 2.1, Adobe Systems, Inc.

SEE ALSO
X(1), Xserver(1), bdftosnf(1)

Glossary

application program
A computer program that performs some useful function, such as word processing or data base management.

application server
A computer used solely to provide processing power for application programs.

active window
The terminal window where what you type appears. If there is no active window, what you type is lost. Only one terminal window can be active at a time.

ampersand (&)
Placed at the end of a command to specify that the client started by the command should be started as a background process. The command can be typed after the command-line prompt or included in a file such as `.x11start` or `.hpwmrc`.

background process
A process that doesn't require the total attention of the computer for operation. Background processing enables the operating system to execute more than one program or command at a time. As a general rule, all clients should be run as background processes.

bitmap
Generally speaking, an array of data bits used for graphic images. Strictly speaking, a pixmap of depth one (capable of 2-color images).

bitmap device
An output device that displays bitmaps. The CRT monitor of your system is a bitmap device.

buffer
An area used for storage.

button
A button on a mouse pointing device. Mouse buttons can be mapped to the keyboard.

button binding
Association of a mouse button operation with a window manager function. For example, pressing button 3 on a window frame displays the system menu.

button mapping
Association of a button number with a physical mouse button.

click
To press *and release* a mouse button. The term comes from the fact that pressing and releasing the buttons of most mice makes a clicking sound.

client
A program written specifically for the X Window System. Some clients make their own windows. Other clients are utility programs.

combined mode
A combination of image and overlay planes in which a single display has a single screen that is a combination of the image and overlay planes.

command-line prompt
A command-line prompt shows that the computer is ready to accept your commands. Each terminal emulation window has a command-line prompt that acts just like the command-line prompt you see on the screen immediately after login. Usually the command-line prompt is either a **$** (for Bourne and Korn shells) or a **%** (for C shells), but it can be modified. One popular modification is to print the current working directory and the history stack number before the **$** or **%**. You can find the command-line

prompt by pressing Return several times. Everytime you press Return, HP-UX prints the prompt.

cut buffer
A buffer (memory area) that holds text that has been deleted from a file.

depth
The number of planes in a set of planes. For example, a set of 12 image planes would have a depth of 12.

diskless cluster
The networking of several systems (SPUs) together to share a common hard disk for storage of data and programs.

display
Strictly speaking, the combination of a keyboard, mouse, and one or more screens that provide input and output services to a system. While "display" is sometimes used to mean just the CRT screen, a display, as defined here, can actually include more than one physical screen.

display server
In the X Window System, the display server is the software that controls the communication between client programs and the display (keyboard, mouse, and screen combination).

double buffering
A term describing the method used by Starbase wherein half of the color planes on a monitor are used to display to the screen and the other half are used to compute and draw the next screen display. This provides smooth motion for animation and it is faster. However, it does reduce the number of colors that are available for display on the screen at one time.

double-click
To press *and release* a mouse button twice in rapid succession.

drag
To press *and hold down* a mouse button while moving the mouse on the desktop (and the pointer on the screen). Typically, dragging is used with menu selecting, moving, and resizing operations.

file server
A computer whose primary task is to control the storage and retrieval of data from hard disks. Any number of other computers can be linked to the file server in order to use it to access data. This means that less storage space is required on the individual computer.

fonts
A font is a style of printed text characters. Times Roman is the font used for most newspaper text; Helvetica is the font used for most newspaper headlines.

foreground process
A process that has the terminal window's attention. When a program is run in a window as a foreground process (as opposed to a background process), the terminal window cannot be used for other operations until the process is terminated.

graphical user interface
A form of communication between people and computers that uses graphics-oriented software such as windows, menus, and icons, to ease the burden of the interaction.

home directory
The directory in which you are placed after you log in. Typically, this is `/users/`*username*, where *username* is your login name. The home directory is where you keep all "your" files.

hotspot
The area of a graphical image used as a pointer or cursor that is defined as the "point" of the pointer or cursor.

hpterm
A type of terminal window, sometimes called a "terminal emulator program" that emulates HP2622 terminals, complete with softkeys. The `hpterm` window is the default window for your X environment.

icon
A small, graphic representation of an object on the root window (typically a terminal window). Objects can be "iconified" (turned into icons) to clear a cluttered workspace and "normalized" (returned to their original

appearance) as needed. Processes executing in an object continue to execute when the object is iconified.

iconify
The act of turning a window into an icon.

image mode
The default screen mode using multiple image planes for a single screen. The number of image planes determines the variety of colors that are available to the screen.

image planes
The primary display planes on a device that supports two sets of planes. The other set of display planes is known as the overlay planes. These two sets of planes are treated as two separate screens in stacked mode and one screen in combined mode.

input device
Any of several pieces of equipment used to give information to the system. Examples are the keyboard, a mouse, or a digitizer tablet.

keyboard binding
Association of a special key press with a window manager function. For example, pressing the special keys Shift Esc displays the system menu of the active window.

label
The text part of an icon.

local access
The ability to run a program on the computer you are currently operating. This is different from remote access, where you run a program on a computer that is physically removed from the one you are operating.

local client
A local client is a program that is running on your `local` computer, the same system that is running your X server.

mask
A graphical image used in conjunction with another graphical element to hide unwanted graphical effects.

matte
A border located just inside the window between the client area and the frame. It is used to create a three-dimensional effect for the frame and window.

menu
A list of selections from which to make a choice. In a graphical user interface such as the X Window System, menus enable you to control the operation of the system.

minimize
To turn a window into an icon. The terms minimize and iconify are interchangeable.

modifier key
A key that, when pressed and held along with another key, changes the meaning of the other key. CTRL, Extend char, and Shift are examples of a modifier key.

mouseless operation
Although a mouse makes it easy to use the X Window System, the mouse is not absolutely necessary. The system can be configured to run from the keyboard alone.

multi-tasking
The ability to execute several programs (tasks) simultaneously on the same computer.

node
An address used by the system. For example, each device on the system has its own node. The system looks there whenever it needs to access the device. A node can also be an address on a network, the location of a system.

non-client
A program that is written to run on a terminal and so must be "fooled" by a terminal emulation window into running in the window environment.

normalize
To change an icon back into its "normal" (original) appearance. The opposite of iconify.

overlay planes
The secondary set of display planes on a device that supports two sets of planes. The other set of display planes is known as the image planes. These two sets of planes are treated as two separate screens.

parent window
A window that causes another window to appear. A window that "owns" other windows.

pixel
Short for "picture element." The individual dots, or components, of a screen. They are arranged in rows and columns and form the images that are displayed on the screen.

pixmap
An array of data bits used for graphics images. Each pixel (picture element) in the map can be several bits deep, resulting in multi-color graphics images.

pointer
Sometimes called the "mouse cursor," the pointer shows the location of the mouse. The pointer's shape depends on its location. In the root window, the pointer is an `X`. On a window frame, the pointer is an arrowhead. Inside the frame, the pointer can be an arrowhead (as when it is inside a clock or load histogram frame) or an `I`-beam (as when it is inside a terminal window).

press
Strictly speaking, to hold down a mouse button or a key. Note that to hold down a mouse button *and move* the mouse is called "dragging."

print server
A computer that controls spooling and other printer operations. This permits a large number of individuals to efficiently share printer resources.

remote access
The ability to run a program on a computer that is physically removed from the one you are currently operating. This is different from local access, where you run a program on the computer that you are operating.

remote client
An X program that is running on a remote system, but the output of the program can be viewed on your terminal.

remote host
A computer physically removed from your own that you can log in to. See chapter 4 for prerequisites for establishing a remote host.

resource
That which controls an element of appearance or behavior. Resources are usually named for the elements they control.

restoring
The act of changing an minimized (iconified) or maximized window back to its regular size. The terms restoring and normalizing are usually interchangeable.

.rhosts
A special file used in network environments that enables a remote user to log into your local system without using a password. Obviously, this has a considerable impact on the security of your system.

root menu
The menu associated with the root window. The root menu enables you to control the behavior of your environment.

root window
The root window is what the "screen" (the flat viewing surface of the terminal) becomes when you start X. To a certain extent, you can think of the root as the screen. The root window is the backdrop of your X environment. Although you can hide the root window under terminal

windows or other graphic objects, you can never position anything behind the root window. All windows and graphic objects appear "stacked" on the root window.

screen
The physical CRT (Cathode Ray Tube) that displays information from the computer.

screen dump
An operation that captures an image from your screen, saves it in a file, and enables you to send that file to a printer for hardcopy reproduction.

server
A program that controls all access to input devices (typically a mouse and a keyboard) and all access to output devices (typically a display screen). It is an interface between application programs you run on your system and the system input and output devices.

stacked mode
A combination of image and overlay planes in which a single display has two "logical" screens, one the image planes, the other the overlay planes. Typically, the image planes are used to display graphics while the overlay planes are used to display text.

system menu
The menu that displays when you press the system menu button on the HP Window Manager window frame. Every window has a system menu that enables you to control the size, shape, and position of the window.

Term0
An HP level 0 terminal. It is a reference standard that defines basic terminal functions. For more information, see *Term0 Reference* in the HP-UX documentation set.

terminal-based program
A program (non-client) written to be run on a terminal (not in a window). Terminal-based programs must be "fooled" by terminal-emulation clients to run on the X Window System.

terminal emulator
A client program that provides a window within which you can run non-client programs. The non-client program runs just as though it were running from a real terminal rather than a window acting as a terminal.

terminal type
The type of terminal attached to your computer. HP-UX uses the terminal type to set the `TERM` environment variable so that it can communicate with the terminal correctly. The terminal type is usually set at login, but can be set afterward.

terminal window
A terminal window is a window that emulates a complete display terminal. Terminal windows are typically used to "fool" non-client programs into believing they are running in their favorite terminal—not a difficult task in most cases. When not running programs or executing operating system commands, terminal windows display the command-line prompt. Two terminal windows are supplied with X11—`hpterm`, which emulates HP terminals, and `xterm`, which emulates DEC and Tektronix terminals.

text cursor
The line-oriented cursor that appears in a terminal window after the command prompt. The term is used to distinguish the cursor used by a window from the cursor used by the mouse, the pointer.

tile
A rectangular area used to cover a surface with a pattern or visual texture. The HP Window Manager supports tiling, enabling users with limited color availability to create new color tiles blended from existing colors.

title bar
The title bar is the rectangular area between the top of the window and the window frame. The title bar contains the title of the window object, usually "Terminal Emulator" for `hpterm` windows, "xclock" for clocks, and "xload" for load histograms.

transient window
A window of short duration such as a dialog box. The window is only displayed for a short time, usually just long enough to get some direction from the user.

window
A data structure that represents all or part of the CRT display screen. It contains a two-dimensional array of 16-bit character data words, a cursor, a set of current attributes, and several flags. Visually, a window is represented as a rectangular subset of the display screen.

window-based program
A client or program written for use with the X Window System. The "opposite" of a window-based program is a terminal-based program.

window decoration
The frame and window control buttons that surround windows managed by the HP Window Manager.

window manager
The window manager controls the size, placement, and operation of windows on the root window. The window manager includes the functional window frames that surround each window object as well as a menu for the root window.

X0.hosts
A file that tells the X Window System which remote hosts can access the local server and hence the local display.

xclock
An X11 client program that displays the time, either analog (hands and dial) or digital (text read out).

xload
An X11 client program that displays the work load of the system as a histogram.

xterm
An X11 client program that displays a terminal window that emulates DEC and Tektronix terminals.

Index

D

E

K

L

M

R

S

T

U

V

W

X

HP Part Number
98794-90001
Microfiche No. 98794-99001
Printed in U.S.A. E0989

For Internal Use Only